10-K and 20-K Technical Guide

December 20, 1985

Updates: June 12, 1986

DUBNER

Dubner Computer Systems, Inc.
6 Forest Avenue
Paramus, New Jersey 07652
(201) 845-8900

A Member of the Grass Valley Group

TABLE OF CONTENTS

SECTION 1: Introduction..1-1

SECTION 2: 10K CPU Board and Circuit Theory....................2-1

Overview...2-1

Discussion of the Motorola 68000...............................2-1

The 10K/20K CPU Board..2-3

The Reset Circuitry..2-3

RAM/PROM Determination...2-4

Prom and Related Circuitry.....................................2-4

CPU/IO Request...2-6

Memory Access Priority...2-6

Memory Cycle Timing..2-7

DTACK Generation...2-8

RUN/STEP Switches..2-11

Buss Error Detection...2-12

68000 INPUT/OUTPUT Strobes.....................................2-13

Serial Communications Channels.................................2-16

Floppy Disk Interface..2-18

SASI/SCSI Buss Communications..................................2-19

Video Timing Signals...2-21

Interrupt Generation and Acknowledge...........................2-23

Dot Matrix Circuitry...2-25

DOTCLK and USDOT Synchronization................................2-25

Dot Matrix Memory Timing.......................................2-26

HOLDOFF Circuitry..2-27

68000 Access to the Dot Matrix Memory..........................2-30

SECTION 3: 10K Display Board Circuit Theory....................3-1
Overview..3-1
CPU Board to Display Board Connections.............3-1
68000 Main Memory Write..........................3-3
68000 Main Memory Read...........................3-5
Main Memory Refresh..............................3-6
General Description of the Video Display Circuit...3-6
Field/Line Video Timing Signals....................3-7
Quick Introduction to AMD 2940 Address Generators..3-7
Overview of the Address Generation Sequence........3-8
Line Pointer Address Circuitry.....................3-9
Data Pointer Address Circuitry....................3-10
Recapitulation of the Start-of-the-Line Logic.....3-15
The 0xxxxxE Exception............................3-16
Normal Line Processing...........................3-17
Ending a Video Line..............................3-19
29521 Control and Clock Signals...................3-19
General Discussion of the Color Palette...........3-19
Switching the Two Palettes........................3-21
Generating Red, Green, Blue, and Key Signals......3-21
Digital-to-Analog Conversion.....................3-22
Cursor Insertion.................................3-22
SECTION 4: CPU Checkout Procedures..........................4-1
Overview..4-1
Set Up the Test Site.............................4-1
Turn on the Power................................4-2
Check the Clock Circuitry........................4-2
Check out the Video Timing Circuitry..............4-2

Refresh Timing and Memory Cycle Timing............4-4
Check Memory Cycle Timing.........................4-5
Test Program "OUTOO"..............................4-7
Initial Serial I/O Test...........................4-9
Test the Dot-Matrix Memory........................4-9
Install Dot-Matrix Optix.........................4-13
CPU Final Test...................................4-13
SECTION 5: Display Checkout Procedures.......................5-1
Overview..5-1
Setting Up the Test Site..........................5-1
Check the Power Supply Voltages...................5-1
Checking out the Basic Memory Functions...........5-1
Check RAS/CAS Timing..............................5-2
Test the Display..................................5-3
Run the Diagnostic Mode Displays..................5-4
SECTION 6: SEK Theory..6-1
The Sync Regenerator..............................6-1
Genlock Reference Input and D.C. Restoration......6-1
Four-Times Subcarrier Phase-Locked Loop...........6-1
Burstgate Generation............................6-2
Defining the Start of a Frame.....................6-5
The RGB to NTSC Encoder...........................6-6
Linear Keyer......................................6-7
SECTION 7: SEK Checkout......................................7-1
Genlock Reference Input...........................7-1
Phase-Locked Loop.................................7-1
IFL Circuitry.....................................7-1
Sync/Blank Generation.............................7-1

RGB Encoder....................................7-2
Linear Keyer....................................7-2
SECTION 8: Connectors....................................8-1
SECTION 9: Equipment Adjustments.........................9-1
Timing Adjustments.......................................9-1
Encoder Section Adjustments..............................9-2
Linear Keyer Adjustments.................................9-3

Welcome to the 10K/20K Theory of Operation and Diagnostic Procedures manual.

This document has been written primarily for the people who are going to be manufacturing and maintaining the Dubner Computer Systems 10K/20K video graphics generator. The 10K and the 20K are identical from an electronics standpoint - the 10K is outfitted with a floppy disk drive, and the 20K has an Iomega removable cartridge drive. The 10K/20K has two or three circuit boards. The first is the CPU board, the second is the DISPLAY board. The optional third board is the SEK board - SEK stands for Sync/Encoder/Keyer.

This document will have two sections for each board: One section describing the circuit theory for that board, and a second section describing the manufacturing and test procedures used at Dubner Computer for checking out and debugging that board. The checkout procedures also contain, a copy of the checklist used at Dubner for making sure the critical signals on the board are okay before the board is shipped in a system. You will also find some appendices which contain information about the Programmable Array Logic parts used in the 10K/20K.

The two sections for the SEK board have not at this time been written.

The circuit theory sections do not make light reading. It is heartily suggested that they be read with a copy of the schematic close at hand and that attempts be made to trace out the various circuits as they are described.

The diagnostic procedures are, if possible, even heavier going than the circuit theory sections. Your best bet is to read them over quickly, and then go to a 10K or 20K and follow the procedures through using an oscilloscope. Out in the field you will have to skip over certain diagnostics which require specific PROM programs or any special hardware jigs that we use at the factory. Our debugging package known as OPTIX will not be generally available either. Nevertheless, you should be able to learn a lot by tracing out and measuring the suggested signals.

I sincerely hope that you are able to go through these descriptions and feel like you have learned something about these machines.

Good luck!

Robert Dubner
Dubner Computer Systems
6 Forest Avenue
Paramus, New Jersey 07652
(201) 845-8900

OVERVIEW

This section is designed to give the reader some idea of what all of the hardware on the CPU board of the 10K/20K is supposed to be doing.

During the course of the discussion of the circuit, I will have many occasions to refer to signal names, many of which are active low. In the drawing active low signals are designated by a 'bar' over them, meaning negation. In writing about them, I indicate such signals by placing asterisks around them, thus the signal ***RESET*** goes low when the signal **RESET** goes high.

DISCUSSION OF THE MOTOROLA 68000

We will first discuss the circuitry of the CPU board. In order to do this, we must first have some understanding of the Motorola 68000 microcomputer chip. A complete understanding can only come from a study of Motorola's liturature, but we can go over the points which are relevent to the 10K.

The 68000 has a 16-bit bi-directional data buss. It has a 23-bit address buss which allows the addressing of 8,388,608 16-bit words. Two other control signals - ***UDS*** and ***LDS*** (Upper Data Strobe and Lower Data Strobe - the asterisks indicate a signal which is active low) - allow individual addressing of the upper and lower bytes of each word. Thus, the 68000 has a 16 megabyte address space. There is a ***WR*** line which indicates that the 68000 will be driving the data bus. The ***AS*** (Address Strobe) line indicates when the address lines contain a valid address.

All 68000 accesses follow the following sequence:

1) The ***WR*** and address lines are established, and ***AS*** is turned on.

2) ***LDS*** and/or ***LDS*** are asserted.

3) The 68000 goes into a waiting mode and sits there until ***DTACK*** becomes active. This signal is Data Transfer Acknowledge, and lets the 68000 know that data is available on a read, or has been taken on a write.

   If a ***DTACK*** never arrives, the CPU will sit and wait forever, unless a ***BERR*** comes in instead. This signal, which means Buss Error, tells the 68000 to execute a special interrupt for handling the special case of non-existant memory.

4) A short while after ***DTACK*** comes in, the 68000 will release ***UDS*** and ***LDS***. A little bit after that, ***AS*** will be released, and the cycle is over.

The 68000 uses memory-mapped I/O - unlike the Intel 8080, there are no explicit input/output instructions. Any I/O accesses which need to be made are done by decoding specific memory addresses and hitting the individual hardware elements accordingly. Consequently, with the exception of interrupts, all 68000 accesses follow the sequence outlined above.

Interrupts are handled only slightly differently. The 10K has only one interrupt, so we will look at that. The hardware requests an interrupt by pulling down the ***IPL2*** line. The 68000 responds by putting the interrupt level on the address lines (which the 10K ignores) and asserting ***AS***. The way you know that this is an interrupt acknowledge is by examining the three control lines **FC2**, **FC1**, and **FC0**. These three lines, which are valid during ***AS***, tell the world what kind of access this is - when all three are high, we are dealing with an interrupt acknowledge. At this time it is up to the hardware requesting the interrupt to put the interrupt vector onto the lower eight bits of the data buss, and hit ***DTACK***.

The only other special case we need to consider is the reset sequence, and then we can plunge into the 10K CPU board. The 68000 reset sequence is started when both ***RESET*** and ***HALT*** are hauled low. When this happens, or rather after they are released, the 68000 burbles and chuckles to itself for a while and then does perfectly normal accesses of the first four words of memory - locations 0000000, 0000002, 0000004, and 0000006.

Let us digress a bit and speak of notation. There will be many occasions when we will have to specify addresses, memory contents, bit configurations, and whatnot. I will use hexadecimal notation almost exclusively. I might use some decimal notation where appropriate, but unfortunately for those individuals who still insist that octal is the only sensible way to do arithmetic, we will be pretending that we have eight fingers on each hand, not four. All hex values will have a "0" prefix, and the rest of the digits will reflect the size of the value being represented. Thus, an 8-bit byte might be 0A2, a 16-bit word could be 05555, and a 32-bit longword would be something like 012345678. The 68000 address space needs 24 bits to address every byte, so a typical address might be 0FFFF10.

Anyway, the 68000 gets its starting vectors from the addresses mentioned above. The first pair of words is the location of the stack, the second pair indicates where the CPU is to commence executing instructions. These addresses are stored as 32-bit longwords, which in the 68000 are stored high-order word first.

THE 10K/20K CPU BOARD

The CPU board has a number of functional elements. These could be listed in a number of different ways, but basically we have:

1) The 68000, and associated timing and control.
2) PROM
3) Four serial I/O channels, each RS-232 or RS-422.
4) The Dot Matrix display.
5) The SCSI buss disk interface for the Iomega.
6) The floppy disk controller.
7) Some timing for the DISPLAY board.

Although we will be going over this again as we discuss the individual elements of the circuits involved, it may be helpful at this time to see how the address space of the 10K is divided up.

0000000-03FFFFF	The four megabyte main display RAM. (Located on the DISPLAY board.)
0400000-0BFFFFF	Nothing.
0C00000-0C7FFFF	PROM.
0C80000-0C9FFFF	Dot Matrix RAM.
0D00000-0D003FF	Palette RAM. (Write only, located on the DISPLAY board.)
0D00400-0D7FFFF	Repeated palette RAM.
0D80000-0FFFEFF	Nothing.
0FFFF00-0FFFFFF	I/O Strobes. (We will discuss these in detail when we get to the I/O section.)

We can now start working our way through the board.

THE RESET CIRCUITRY.

The hardware reset for the 68000 and the rest of the circuitry is generated by the circuit in the upper-left-hand corner of sheet 2 of the CPU schematic. On initial power-up, or when the RESET button is pressed, the 555 timer is fired off and the *RESET* and *HALT* lines are held down for something over one second. Some elements of the circuitry require a positive-going reset pulse, so *RESET* is inverted to generate the **RESET** signal. These signals are used to clear the 68000 and any other conditions which might otherwise come up in an amibiguous state.

RAM/PROM DETERMINATION.

The more discerning among you will have noticed in the table above that memory locations 0000000-0000007 are to be found in the RAM on the DISPLAY board. On reset, the 68000 is going to try to access those locations to find its starting stack and program counter values. Pulling these values out of RAM, particularly on power-up, is an exercise of dubious value. On sheet 1 of the schematic, at the top middle you will find a pair of NAND gates organised as a set/reset flip-flop. The purpose of this circuit is to force the first four memory access to come out of PROM.

Looking at this circuit, you will see that when address bits 23 and 22 are both low, the RAM enable line ***RAMEN*** will go active, except in the following two cases:

1) When ***RAMEN*** is suppressed by an interrupt acknowledge, as indicated by ***INTA***.

2) When ***RAMEN*** is disabled, and ***PROMEN*** is forced, for the first few memory accesses after a RESET pulse. The flip-flop which forces this condition is cleared when an honest access to PROM is decoded.

The sequence of events here is as follows:

1) A reset pulse occurs, setting the RAM/PROM Determination flip-flop.

2) The CPU attempts to access location 0000000, followed by 0000002, 0000004, and 0000006.

3) Because ***RAMEN*** is suppressed, and ***PROMEN*** is active, these accesses will cause data from the first 8 bytes of the PROM to be placed on the data buss. (See PROM circuitry, below.)

4) The initial value for the program counter, which the 68000 will read when it accesses locations 0000004-0000007, will be an address which is legitimately in the PROM address space. When that address is decoded the signal ***PROMADD*** will go low, which clears the RAM/PROM Determination flip-flop.

5) After that, address determination continues in an ordinary fashion.

PROM AND RELATED CIRCUITRY.

In the lower-left-hand corner of sheet 1 of the CPU schematic you will find a circuit block labeled "PROMs and PROM address decode."

There are four PROM sockets on the CPU board. These are arranged as an upper bank of 16-bit words, and a lower bank. The purpose of all those jumpers is to make it possible to use a wide variety of PROMs. It is possible to have as little as 4K of memory by using two inexpensive 2716s, or as much as 128K of memory by using four rather costly 27256s. We feel it likely that the 10K/20K product will be shipped with two 2732s containing only a disk boot program, and that the larger PROMs will only be used in debugging at the factory. In any event, the jumper settings are explained by the table on the schematic. A more complete understanding of just why those signals are routed the way they are can be had by examining Intel's (to name just one manufacturer) data sheets, particularly the ones which spell out the differences between the 2716, the 2732, the 2764, the 27128, and the 27256 PROMs.

Let's assume that the jumpers are set up for the PROM of your choice. We can now look at some of the other circuitry. The F138 (which is a 3-line to 8-line decoder) is in charge of chopping the uppermost four megabytes of the address space into eight one-half megabyte pieces. Notice that for this decoder to function, address bits 23 and 22 both have to be high, ***AS*** must be low (signifying a valid address) and **INTA** must be low (indicating that this is not an interrupt acknowledge cycle). As detailed above, the first piece is PROM, the second is the Dot Matrix display, and the third is the palette RAM. The other five one-half megabyte chunks are not used. The decodes for the Dot Matrix and the palette are called ***DISPADD*** and ***PALADD*** and will be examined more fully later. For now, we will examine what happens during an access into PROM.

It is really pretty simple - the 68000 sends out a valid address in the PROM space, which makes the ***PROMADD*** line of the decoder go low. (This ***PROMADD*** signal is what clears the flip-flop in the RAM/PROM Determination circuitry.) The ***PROMADD*** signal is gated together with the ***PROMEN*** signal from the RAM/PROM Determination circuit to create the ***PROM*** signal. This signal is used to generate ***PRLO*** and ***PRHI***, which determine whether the upper pair or lower pair of PROMs is being accessed. The ***PROM*** signal also enables the pair of drivers that isolate the PROMS from the data buss.

Okay, we are about halfway through the PROM access cycle. The 68000 has sent out the addresses and the ***AS***. The decoder has figured out that this is a PROM access and has enabled the correct pair of PROMs. The PROMs themselves are hanging on the address buss and are sending the correct data to the AS573 drivers, which in turn have been enabled by the ***PROM*** signal and are placing that data onto the main data buss. All that's needed now is somebody to give the 68000 a ***DTACK***, so that the 68000 knows that the data is there.

Unfortunately, in order to understand how the ***DTACK*** gets generated, we have to wade our way through what may well be the most complicated circuitry on the board.

Let's go through it, step by step. We will follow a 68000 access through from start to finish.

CPU/IO MEMORY REQUEST.

On sheet 2 of the schematic, on the left-hand side, just below the reset circuitry, you will find a flip-flop that gets clocked when **AS** goes high - in other words, when the 68000 announces that it wants to do some kind of access. When this occurs, the signal *REQCPU* goes low, indicating that the 68000 is requesting control of the hardware.

MEMORY ACCESS PRIORITY.

Wait just a minute - who else can get control of the hardware, I hear you ask. The answer is that the display hardware on the DISPLAY board also generates memory accesses - when a picture is sufficiently complicated it can require every other memory cycle. The display memory also needs to be refreshed every now and then, during which time the 68000 can't get in.

Later on, you will discover that the DISPLAY board has its own data and address busses which are isolated from the CPU board. Consequently, it is in principle possible for the CPU board to be accessing PROM or the I/O ports while the DISPLAY board does its own accesses without one having to wait for the other. As a matter of fact, we make use of this principle by allowing a 68000 cycle to take a lot more time then a simple memory cycle (the serial I/O hardware, for instance, is very slow.) But the beginning of every cycle is treated as if it were a main memory cycle. The 68000 program for the 10K will live in display memory, PROM will only be used for booting, and you almost never access I/O ports. Thus, in normal operation, virtually every cycle will be a main memory cycle. Only occasionally will the 68000 want to get at something besides memory, so in terms of timing, we will treat every 68000 access as if it were trying to get at memory. In fact, every 68000 access will cause a main memory access - the decision to do a write or put the read data onto the data buss will be made at the appropriate point in the memory cycle by evaluating ***RAMEN***.

I told you it was complicated.

We were discussing Memory Access Priority. We have seen that there are three pieces of hardware that can want to get at the memory - they are the 68000, the memory refresh hardware, and the DISPLAY board itself. These three requests are sorted out by the circuit immediately to the right of the ***REQCPU*** flip-flop.

The highest priority request is *REQTOP*. This is the request from the DISPLAY board - its name is a holdover from an early design stage when we thought we might have a two-plane device, with a top display plane and a bottom display plane. The middle priority is *REQREF*, which is from the refresh circuitry. The lowest priority is the 68000 request, which is gated with CPUCLK in order to line things up properly. We probably should digress and talk about some of these clocks for a moment.

The clock circuitry at the top of sheet 2 generates three fundamental clocks - CLK (and *CLK*), which run continuously at 25MHz; CPUCLK, which runs at 12.5MHz and is used to drive the 68000 clock line; and DOTCLK, which runs at the same frequency as CLK but is synchronized to the incoming video timing signals by the HOLDOFF circuitry (which we will examine later.) The memory access priority circuit is driven by *CLK* which runs twice as fast as CPUCLK. Thus, it is possible for a 68000 memory cycle to start at two different places relative to the CPUCLK. One of those possibilities is too late - it causes the 68000 to drop *AS* after the next cycle has begun. Gating *REQCPU* with CPUCLK prevents that late-ending cycle.

Back to the priority decode. The F379 grabs the various requests on the rising edge of *CLK* provided that *DE7* is low. Once the requests are grabbed, the assortment of gates on the F379's outputs sort out who is the lucky winner. The 5-input NOR gate at the bottom is in charge of letting the Memory Cycle Timing circuitry know that it is time to go on a new cycle.

MEMORY CYCLE TIMING.

The basic philosophy of this circuit is that a memory cycle consists of eight CLK cycles - in this machine that is 320 nanoseconds. Those eight cycles are named DE0, DE1, DE2, DE3, DE4, DE5, DE6, and DE7. The current state is determined by the four-bit ring counter which is generated by PAL3 in conjunction with the AS574 which follows it. (If you are not familiar with PALs, you will learn some more about them when you get to the *DTACK* discussion.) The bit configuration of each DE is found in the little table on the schematic.

This circuit will bear some little examination, but the primary thing to keep in mind is that it "holds" in DE7 when it otherwise has nothing to do. Consider what happens when you come to the end of a memory cycle and there are, for the moment, no requests pending. (This happens all of the time, since the 68000 does not use up every possible memory access.)

The ring counter is merrily counting along, DE3, DE4, DE5. When we reach DE6, which starts on the rising edge of CLK, *PRIEN* goes low, which enables the F379 in the Memory Access Priority circuit. When CLK goes high again (this will be at the beginning of DE7) all of the Q outputs of that F379 will go high, since there are no requests pending. Likewise, all of the Q-not outputs will go low.

Since all of the Q-not outputs are low, all of the inputs of the S260 are low, which means its output will go high. PAL3 is designed so that if the output of the S260 is high, then the circuit stays in the DE7 state forever.

Or rather, until the next request comes along. The PAL3 holds ***PRIEN*** low as long as the output of the S260 is high and the ring counter is in DE7. This means that the priority decode F379 is clocking every **CLK**. When any request comes in, one of the Q-not outputs will go high, so the output of the S260 will go low. This causes the PAL3 to let *PRIEN* go high, and lets go of the ring counter hold, so on the next rising edge of **CLK** we will be in DE0.

The various gates in the Memory Cycle Timing circuitry decode an assortment of signals that are used here and there for driving drivers, clocking latches, and generating ***DTACK*** for the different circuit elements. Many of these will be encountered as we progress through the rest of the circuitry.

Two deserve special mention, since they are of such prime importance. The pair of NOR gates in the RAS/CAS Generation box generate the Row Address Strobe and the Column Address Strobe for the memory on the DISPLAY board. Note that ***RAS*** comes down at DE1, and ***CAS*** 40 nanoseconds later. An examination of the timing will show that there is ample time during a read access for even 150nsec DRAMs, even with 30 nanoseconds or so of propogation delay through the assortment of drivers and transceivers between the RAM chips and the 68000 chip.

Two of those signals, ***DE3*** and ***DE5*** go down to the *DTACK* generation circuitry where they are used to help decide when to tell the 68000 that it can finish up the current cycle. You will recall that we left the 68000 in the middle of a PROM access a couple of pages back - let's see if we can finish it up.

DTACK GENERATION

In the lower-left-hand corner of sheet 2 of the schematic you will find the circuitry which is responsible for generating *DTACK* most of the time. The heart of the circuit is a 16L8 PAL. Those of you who are familiar with this class of chips should go get a cup of coffee while I introduce them to the newcomers.

PAL means Programmable Array Logic - the parts that we are using are made using bipolar transistor technology. The 16L8 is a very basic and ordinary kind of PAL - each tri-state output is driven by a 7-input NOR gate. Each input to each NOR gate is driven by a 32-input AND gate. The decision as to whether or not each output is tri-state is made by another 32-input AND gate. Thus the internal guts of a 16L8 consist of eight 7-input NOR gates being driven by sixty-four 32-input AND gates.

For further information I commend to you Monolithic Memories's PAL Handbook, or perhaps the similar information that you might get from Advanced Micro Devices. The point is that each and every AND gate can be connected to any one of 16 I/O pins (or its inversion) by programming the array of 2,048 fuses that this device has. Hence, Programmable Array Logic. Simple, don't you see?

Programming these little buzzards by hand is tedious, time-consuming, and error prone, so a number of programs have been written which accept Boolean logic, truth tables, what-have-you and from them generate the fuse maps for the PALs. We happen to be using ABEL, which is DATA I/O Corporation's version. The 10K uses a number of PAL's - in addition to explaining them in the text we will be including the ABEL source files and the resulting outputs.

Where were we? Oh, yes - I was going to explain what the PAL in the DTACK generation circuitry is supposed to do. If you look at the input pins on the left side of that PAL you will see only one signal that we have not already explained. That signal is ***CPUCY***. If you look near the center of the drawing you will see a box labeled Read/Write Pulse Generation. We will be looking at it again later, when we examine the floppy disk and serial I/O circuits, but for now notice that the flip-flop, which is clocked near the beginning of each memory cycle by ***DEO***, generates ***CPUCY*** which goes low if the cycle which has just started has been allocated to the 68000. ***CPUCY*** goes away when ***DTACK*** comes along and kills it or when the next memory cycle starts, whichever comes first. In other words, ***CPUCY*** is similar to ***GRTCPU***, but with slightly different implications.

Anyway, back to the PAL. (The following discussion may be subject to change as time goes on - that's one of the beautiful things about using PAL's. To understand exactly what the PAL is doing, you need to look at the latest revision to the logic inside the PAL.) This PAL has a number of functions.

1) If any of ***DEUCE1***, ***DEUCE2***, or ***FLOPPY*** go low, then ***DEUFL*** goes low. This signal is used in the serial I/O and floppy disk circuitry.

2) When a 68000 cycle gets started which uses either of the DEUCE chips or the floppy disk controller chip, pin 18 of the PAL goes low which starts the F163 counting. The four output pins of the F164 go back into the PAL. (Even though those PAL pins are on the right side, in this case they are configured as inputs - more PAL magic.) When the count gets up to the correct value, the PAL will cause a ***DTACK*** to be generated. The reason for this is because the DEUCE chips and the floppy disk controller chip are relatively sluggish, and need fairly long read/write pulses and ridiculously long address/data setup and hold times. See the write-ups for

that circuitry for specifics.

3) General DTACK generation. During a *CPUCY*, pin 19 will go low (triggering a *DTACK*) when the following conditions are met:

   a) It is an ordinary *IO* and we have reached DE3.
   b) It is a *DEUCE1*, *DEUCE2*, or *FLOPPY* access and we have reached the proper count on the F163.
   c) It is a *PROM* access and we have reached DE5.
   d) It is a *RAMEN* access and we have reached DE3.
   e) It is a *PALADD* access and we have reached DE3.

Actually, while I have been talking about triggering a *DTACK*, what I have really meant was that I was triggering a *DTACKOC*. When pin 19 of the PAL goes low it sets the set/reset flip-flop which causes the open-collector inverter to take *DTACKOC* low. (The open-collector part is used because there is at least one other place on the board that generates *DTACKOC*.) This will, at last, cause *DTACK* to go low at the 68000, provided, of course, that the RUN/STEP circuitry is allowing *DTACK* to occur. (More on this in a minute.) A couple of clock cycles after the 68000 gets the *DTACK* it will release the address strobe, which means the AS signal will go low, which will clear both of the set/reset flip-flops in the *DTACK* Generation circuit.

I think it is now time to review our PROM memory cycle from start to finish.

1) The 68000 asserts *AS*, telling the world that it needs to do a memory access. This sets the CPU/IO Memory Request flip-flop which asserts *REQCPU*.

2) The Memory Access Priority circuit repeatedly latches up all three of the requests when there is no memory access in progress. When it is the 68000's turn, *GRTCPU* will go low.

3) At the same time, the output of the S260 will go low, which fires off the Memory Cycle Timing ring counter. Once started, this counter will progress from DE0 back to DE7. The various decodes on the ring counter allow the rest of the hardware to figure out just where in a memory cycle the machine is.

4) The very first such decode, *DE0* is used to latch up the state of *GRTCPU* to establish that this is a 68000 cycle. This is indicated by *CPUCY* going low.

5) Meanwhile, the PROMs and PROM Address Decode circuitry has been figuring out that this is, indeed, a PROM access. The correct pair of PROMs has been enabled, and the AS573 drivers a putting that data onto the 16-bit data buss.

6) The data from the PROMs is on the data buss, and all is ready to go. Over in the *DTACK* Generation circuity the PAL has noticed that this is a 68000 cycle (since *CPUCY* is low) and that it is a PROM access (since *PROM* is low.) When *DE5* goes low, pin 19 of the PAL will go low.

7) This will cause the set/reset flip-flop to make *DTACKOC* go low, which in turn goes through the OR gate and makes *DTACK* go low, letting the 68000 know that the data is available. Once the data is accepted, the 68000 releases the address strobe, which clears the set/reset flip-flop.

I think that about covers it. We are now mostly done with the confusing stuff, except possibly for the Dot Matrix hardware. But before continuing with the I/O circuitry, we ought to take a look at the RUN/STEP Switches.

RUN/STEP SWITCHES.

We anticipate that the only thing that the STEP switch will be used for is debugging the CPU board. All of the hardware on the CPU board can be accessed in single-step mode: The PROM, the Dot Matrix RAM, the Serial I/O and the floppy and SCSI disk stuff. You can't get at the RAM on the CPU board - we feel that this is no big deal, since if you have the two boards connected together there is no good way to look at the address and data busses in any event. To repeat, it is our feeling that single-step mode is provided to make it easier to get the PROM working - once you've done that, you use PROM programs to get the rest of the hardware up.

With that caveat, let's look at the RUN/STEP stuff, which is found in the middle of the left-hand side of sheet 2 of the schematic.

Notice first that there are just the two push-buttons. Normally the machine is in the RUN mode - you can see that either pushing the button or a *RESET* pulse will cause the machine to go into RUN. Being in RUN means that the upper F74 is held in the set state, which means that the output running to the *DTACK* Generation circuit is low, which in turn means that *DTACK* will go low with *DTACKOC*.

Pushing the STEP button takes the machine out of RUN and puts it into the single-step mode. The first thing that happens is that the *BERR* signal is held high - the buss error stuff is disabled. More about this later. The other thing that happens in STEP mode is that each *AS*, when it goes away, causes that upper F74 to be cleared. This causes the signal going to the *DTACK* OR gate to go high, which will prevent the following *DTACK* from going low. This will cause the 68000 to go into, and stay in, a wait state. This wait state will continue until the STEP button is pressed again.

When the STEP button is pressed, the debouncing one-shot will trigger (CLK will continue to retrigger it until the button is released. The rising edge of the one-shot's positive-going output clocks the upper F74 into the set state. This will allow the *DTACK* to occur.

Let us review, step-by-step, what happens when the 68000 board is in a wait state.

1) The 68000 is in a wait state. This means:
    a) *AS* is low, since we are in the middle of a cycle.
    b) *DTACKOC* is low, since whoever was accessed is trying to complete the cycle.
    c) The upper F74 is in a cleared state, preventing the *DTACK*.

2) Somebody presses the STEP button.

3) The upper one-shot fires, which sets the upper F74.

4) This allows the *DTACK* to go through.

5) Once the 68000 realizes that *DTACK* has occurred, it will allow *AS* to go high.

6) This causes the upper F74 to be cleared. (Recall that it also causes *DTACK* to go away.)

7) The 68000 starts its next memory cycle, which gets halted at the 'waiting-for-*DTACK*' stage because the upper F74 is cleared. This brings us back to step 1).

Once you have gotten bored with watching the 68000 sit around in single-step mode doing not much of anything, you can send it back into RUN mode by pressing the RUN button. This does two things:

1) It clears the set/reset flip-flop (which is made up of those two ASL00s, don't you know) which in turn holds the upper F74 in the set state, which in turn allows *DTACKOC* to always become *DTACK*.

2) It forces the buss error circuitry into a 'no-error' state.

I suppose it is time to talk a bit about buss errors.

BUSS ERROR DETECTION. (It is in the RUN/STEP block.)

A buss error is a special form of error processing in the 68000 reserved for handling the problem of accessing a non-existing memory segment, or, What Does One Do When One Never Gets a *DTACK*? The answer, my friend, is you give it a buss error instead.

As we have seen, in single-step mode the buss error circuitry is disabled. In RUN mode, it is very much enabled. The lower one-shot is constantly being retriggered by *AS* going low once or twice a microsecond. Even the longest instructions (the worst-case multiply, for instance) only last five or ten microseconds. That one-shot is set for about 50 or 60 microseconds.

If a memory location is accessed for which there is no hardware willing to generate a *DTACK*, *AS* will go low and stay low. When that one-shot finishes its cycle, the inverted ouput will go high, clocking the flip-flop. Clocking that flip-flop will cause its inverted output to go low, which will in turn generate a buss error.

The 68000 looks for the *BERR* signal the same way that it looks for *DTACK*. Once it sees the *BERR* it will think to itself for a moment and then release *AS*. When AS goes low it clears the *BERR* flip-flop, which gets rid of the buss error signal.

68000 INPUT/OUTPUT STROBES.

Believe it or not, we have actually moved pretty far along in our understanding of how this CPU board really works. You may think that you are completely confused, but hopefully you can now start to look at this schematic with a certain amount of knowledge, however fractured, of just what might be going on. At this stage we have a functioning 68000 computer which can read stuff from its PROMS. We can reset it, and single step. We understand the basic timing of a memory access cycle, and we have a fair idea of what kinds of signals the 68000 generates and expects in return.

It is now time to start looking at the various things that the 68000 can do. We will start with its I/O strobes.

In the center of sheet 1 of the schematic you will find a block labeled Input/Output Strobe Decode. As explained earlier the 68000 has no specific I/O instructions. The concept is useful, however, so the very uppermost 256 bytes of the memory space - locations 0FFFF00 through 0FFFFFF - have been arbitrarily designated the I/O space. The major decode is done by the ALS677, whose job is to bring down the *IO* signal whenever the upper 16 bits of the address buss are high (provided this is not an interrupt acknowledge cycle.) When *IO* is low, the F138 chops the first 128 bytes of the I/O space into eight pieces, only four of which are used.

As can be seen from the schematic, the four major I/O addressing chunks are as follows:

0FFFF00-0FFFF0E	Random I/O.
0FFFF10-0FFFF1E	First DEUCE. (First two serial lines.)
0FFFF20-0FFFF2E	Floppy disk controller.
0FFFF30-0FFFF3E	Second DEUCE. (Other two serial lines.)

We will examine the DEUCEs and the floppy disk controller later. For now, we will tabulate and explain the individual random I/O strobes. On the drawing, and in this discussion, I use the shorthand notation ***OUT00*** instead of "output location 0FFFF00".

We will examine the output strobes first. You will notice that even when the machine is in single-step mode the output strobes are short pulses. We did this when we discovered that the 68000 is very skimpy in the amount of time it keeps the data around after letting go of ***UDS*** and ***LDS***. The various stroves are as follows:

OUT00 "SERIALIO"

> This strobe latches up the LS175 on sheet 3 of the schematic. If data bit 3 is high, it means that the first serial I/O channel is RS-422. Otherwise, that channel is RS-232. Data bits 2, 1, and 0 do the same for the second, third, and fourth serial channels.

OUT02 "INTERUPT"

> As we will be seeing later, this computer has a single interrupt which can be caused by either one of two things - 1) the video circuitry has detected that we are at the start of a field, and 2) we have counted up a certain number of lines. In other words, we get a interrupt every N lines. ***OUT02*** establishes what N is.

OUT04 "POINTER"

> As we shall see when we examine the DISPLAY board, ***OUT04*** is used to initialize the line-pointer-table circuitry.

OUT06 "POINTER"
OUT08 "POINTER"

> This 32-bit value is the location of the beginning of the line-pointer-table. Again, you will learn all about this, in horrifying and excruciating detail, when we cover the DISPLAY board.

OUT0A "SASIDATA"

> This word, or rather the bottom eight bits of it, establishes what information is going out on the eight data lines of the SCSI interface cable. This output strobe also activates the SCSI interface ACKNOWLEDGE signal. (For more information, see the SASI Bus Communications explanation.)

OUT0C "SASISTAT"

> Certain bits of this word establish SCSI interface control signals:

Bit 15 is the RESET line.
Bit 14 is the SELECT line.

Bit 12 is used to make the floppy disk interface single density rather than double density.

OUT0E "PALETTE"

Again, look the the DISPLAY board for a full explanation. Turning on bit 0 of this word will cause the palettes to switch at the top of the next field.

We can now examine the input strobes. Unlike the output strobes, the input strobes go low and stay low throughout the entire 68000 input cycle. They are:

IN00 "UARTSTAT"

Much ado here about but a single bit. Bit 7 of this word, when it is low, indicates that at least one of the four SERIAL I/O channels has a character ready from its input line. We put to moinimize the amount of time spent on the interrupt level deciding whether or not any input is ready.

IN02 "INTERUPT"

This gives you certain useful information about the interrupt you presumably are servicing. Bit 0 tells you if the interrupt was caused by a FIELD pulse. If it is off, then the interrupt was caused by the N-line countdown - more on this later. Bit 1 tells you which of the two video fields you are in.

IN04 "Ser. #"

This number should more properly be called the "Software Identification Number", or SIN. It will be used to identify which machine this is. Various different software options are going to be available, and this number will be used to make sure that a user can only do those functions that have been paid for. 10K and 20K units will each have their own series of numbers, so the software ID will have no relationship to either the CPU board serial number or the system serial number.

To foil casual duplication of the number, the SIN is kept in a PAL whose security fuse has been blown, making duplication impossible.

Extracting the number from the PAL is a little peculiar. If you do a word access of location 0FFFF04, the high-order 8 bits will be the high-order SIN. When you do a byte access of the same location, the low-order 8 bits will be the low-order SIN.

IN06 Not used.

IN08 Not used.

IN0A "SASIDATA"

This places the eight data bits of the SCSI interface onto bits 7-0 of the data buss. It also activates the SCSI ACKNOWLEDGE signal.

IN0C "SASISTAT"

This reads in the SCSI interface status bits:

Bit 15 is the REQUEST line.
Bit 14 is the BUSY line.
Bit 13 is the MSG line.
Bit 12 is the COMMAND/DATA line.
Bit 11 is the I/O line.
Bit 08 is the parity error accumulation bit.

IN0E Not used.

Let me reemphasize that many of these signals will be explained in more detail when we go over the specific circuitry involved.

SERIAL COMMUNICATIONS CHANNELS.

If you turn to sheet 3 of the CPU schematic, you will find the hardware which allows communications on four serial lines. I am not going to go too crazy explaining the Western Digital DEUCE parts - you should look examine the specifications for the WD2123 DEUCE in the Western Digital Corporation Communication Products Handbook for detailed information. I will cover some of the highlights.

Most significantly from a hardware standpoint are the setup, pulse width, and hold times with regard to the ***RDPULSE*** and ***WRPULSE*** signals. An examination of the specifications shows that CS3, CS2, CS1, and C/D must all be stable for at least 50 nanoseconds before either ***RDPULSE*** or ***WRPULSE*** goes low. The specs also indicate that those pulses have to last at least 230 nanoseconds, and that after the pulse is over, those control lines must stay stable for at least 50 nanoseconds afterword. Further, on a write into one of the DEUCEs, the data must stay stable for at least 100 nanoseconds. For a CPU as nimble as the 68000, these times are almost unconscionably lengthy.

Well, there is no getting around them - those times have to be satisfied. On our board, we do so with the Read/Write Pulse Generation circuit in the middle of sheet 2. You will recall seeing that circuit when we examined the ***CPUCY*** signal earlier on. You should now notice the two OR gates which combine the

CPUCY signal with both WR and *WR1* to generate the *WRPULSE* and *RDPULSE* signals.

You will note that whichever pulse is going to be activated comes down roughly at the end of *DEO*. If you go back over the Memory Timing circuit you will find that DEO ends about 60 nanoseconds after *CPUGRT* goes low. And that in turn is at the very least a dozen or so nanoseconds after the 68000 generated its addresses and *AS*. So there is no question that the setup time will be satisfied.

The length of the read or write pulse is dependent on the PAL in the *DTACK* Generation circuitry. That PAL is set up so that *DTACK* does not get generated until 240 nanoseconds have gone by. *DTACK* is what turns off the pulses, so the 230 nanosecond constraint is met.

Once the 68000 notices that *DTACK* is around, the address information hangs around for at least 80 nanoseconds (satisfying the control signal hold time), and on a write, the data hangs around for about 120, which satisfies the data hold time.

If you are interested in what those reads and writes actually do, I again refer you to Western Digital's documentation. That along with the little table showing which strobes do what should give you the information that you need.

The last point of interest before we leave the DEUCEs should be an introduction to the I/O data buss, which connects to the main data buss through that AS646. The AS646 is a bi-directional latching buss transceiver. On a write to the DEUCEs or the floppy controller it is left transparent, since as we discovered above there is plenty of time to do a write. A read, however, is another story.

These DEUCEs, as we saw, are pretty sluggish. Some 200 nanoseconds after *RDPULSE* starts they will will be so kind as place some data on their output pins. Our trouble starts when we examine the 68000 specifications and discover that the 68000 looks at input data on a read at about the same time that it stops driving the address buss. There is no way that we can satisy both the DEUCE and the 68000 without latching up the data.

This is done by bringing the *CPUCY* signal into the CBA clock of the AS646. *CPUCY* goes up after 240 nanoseconds, which is greater than the 200 data latency delay of the DEUCE. The AS646 then continues to drive the data buss going to the 68000 while the DEUCE, its work over, proceeds to go back to sleep.

Although it was mentioned up above, let's look again at the ***DEUCERDY*** signal. Each DEUCE has two serial channels, each of which has its own status word. If you use those status words to determine if there is an input character ready, you have to do four separate inputs. When transmitting at 9600 baud, you really want to check those status bits every 500 microseconds or so. Although this would not represent a real big load on the computer, we decided to reduce it even further - it was a freebie, since all the hardware was already on the board.

The DEUCE brings out the "input ready" status bits as hardware lines. By putting them all together with the S260 we create the ***DEUCERDY*** signal, which goes down if any of the input status bits are high. This is brought into a buffer which is activated by ***IN00***. (Sheet 1, lower right.) This means the 68000 can determine whether or not further action is needed with only one strobe - not four.

This brings us to the actual receive and transmit hardware. Each serial channel has a 9-pin DE9-S socket associated with it. There are two GROUND lines, two lines which are dedicated to RS-232 communications, and four lines dedicated to RS-422 signals. Transmission in either mode is simple - if you look at the circuitry, you will notice that both types of transmission are always being made. Reception is slightly more complex, since you have to decide which of the two you want to listen to.

There is a 1489 chip which looks at the incoming RS232 lines, and a couple of 8820's which look at the RS-422 signals. Both of these chips generate TTL outputs, which go into the home-grown AND-OR array made up of the two LS08s and the LS32. The control of whether each line is RS-232 or RS-422 is determined by the LS175 which is loaded up by the 68000 using ***OUT00***.

And that, friends, is the serial I/O.

FLOPPY DISK INTERFACE.

Please be so kind as to turn to sheet 4 of the schematic.

Believe it or not, there is almost nothing at all to say about the Floppy Disk Interface. Of the handful of chips there, there is almost nothing of any interest. The oscillator is an oscillator, the LS161 is there purely to divide the clock down to the value needed by the WD2793, the F240 is serving purely as a driver, as is the 7445, and since the 10K uses one and only one floppy disk drive, the LS175 only functions to turn the drive's motor on and select which side of the disk we wish to access.

The configuration of the 34-pin ribbon connector is established by the pinouts of the Shugart 475 5.25" disk drive. All of the circuitry that surrounds the WD2793 is copied out of Western Digital's Storage Management Products Handbook, as is the setup procedure. For further information, I recommend that you read those specifications. Don't expect to do it quickly. We will obviously have to explore some of the functions of the WD2973 later on when we discuss diagnostic and adjustment procedures.

What is relevant to the current discussion is the read/write timing into the WD2793. Happily, the same timing that controls the DEUCEs is applicable here - control setup time is 50 nanoseconds, read/write pulse width is minimum 200 nanoseconds. The control hold time is 10 nanoseconds. Worst case read data latency is 200 nanoseconds, and the data hold time on a write is 50 nanoseconds. Thus, we see that all of the times here are equal to or better than the times for the DEUCE, so we are able to hang the WD2793 onto the same 8-bit buss that we use for the DEUCEs, and we can use exactly the same timing.

That's the good news. The bad news is that we now have to look at the SCSI interface.

SASI BUSS COMMUNICATIONS.

SASI or SCSI - it is all the same thing. SASI means "Shugart Associates Standard Interface", and SCSI means "Small Computer Standard Interface". The 20K uses an Iomega half-height 10 megabyte cartridge disk drive, so I am afraid that once again I have to refer you to a manufacturer's manual - in this case the Alpha-10 OEM Disk Storage Subsystem Technical Description Manual. Once you have waded through that (another task not for the faint of heart) you will better be able to understand the condensed description of an SCSI interface transfer that I am about to walk through.

1) We start in a idle condition - all of the SCSI control lines, both coming and going (as indicated by the little arrows to the right of the connector) are in the high state.

2) Suddenly, without warning, the 68000 decides to access the disk drive. It starts by doing an *INOC* in order to check the *BUSY* line. It should be high.

   If it were low, this would be an error condition. The 68000 would attempt to remedy the situation by setting bit 15 with an *OUTOC*. This would bring low the SCSI *RESET* line, which would presumably clear out the controller.

3) Since the *BUSY* line is high, we can proceed with the select sequence. The next thing the 68000 does is an *OUTOA* with the proper controller address code - for a single-drive system this will be an 001. If there is more than one controller, the others will be numbered 002, 004,

008, etc.; one bit per controller.

This byte is latched up by the AS648. This is a bi-directional, latching transceiver that inverts the busses - necessary because the SCSI data bus is upside down.

Note that because the SCSI controller is in the idle state, the *I/O* signal is high. A glance at the specs for the AS648 will show you that this causes the SCSI side of the transceiver to be driven constantly with the latched data.

4) Once the controller address has been placed on the SCSI data lines, the 68000 executes an ***OUTOC*** with a 1 in bit 14. This causes the ***SELECT*** bit of the SCSI interface to go low.

5) The 68000 now sits and waits for the SCSI controller to respond, which it does by pulling ***BUSY*** low. The 68000 looks at the condition of that bit by doing an ***INOC*** and checking bit 14.

6) Once ***BUSY*** goes low, the 68000 does another ***OUTOC*** and releases the ***SELECT*** line.

Time to digress for a moment and explain SCSI data transfers. In the SCSI format, the disk controller (the 68000 is the host) is a very demanding peripheral. It is he who decides which way data is going to flow (*I/O*), it is he who decides whether it is data or command information (*C/D*), and it is he who decides exactly when the transfer takes place (***REQUEST***). **On transfers to the disk, he pulls** *REQUEST* low, which means "Okay buddy, I am waiting for some data." The host responds by putting the data onto the eight data lines and then yanking down the ***ACK*** line. On a transfer from the disk, the disk puts the data onto the eight data lines and pulls down ***REQUEST*** in order to announce "Here's your data". The host responds with ***ACK*** in order to tell the disk, "I got it".

7) The 68000 is now waiting to send the SCSI controller command information. It waits for the ***REQUEST*** line to go low - checking it by doing ***INOC*** and looking at bit 15. Once bit 15 goes low he checks the *I/O* and *C/D* lines to make sure they are valid for a command transfer. (*I/O* should be high and *C/D* should be low. If, for instance, they are both low, the controller is trying to announce some kind of error, most likely that it is currently busy and that you should come back later.) If they are valid for a command transfer, the 68000 can start transferring command bytes.

8) The 68000 sends the first command byte to *OUTOA*. This does two things - it places the byte on the data lines, and it clocks the F74 in the middle of the circuit. Notice that the flip-flop can be clocked - its set line is not being activated because the *REQUEST* line is low.

   Once that flip-flop is clocked, the SCSI *ACK* line will go low. This lets the disk know that the data is ready for it to grab.

9) When the controller has grabbed the data, it will release the *REQUEST* line. Once the *REQUEST* line goes high it will simultaneously let the *ACK* line go high and set the middle flip-flop. This puts the circuitry in a state where once again the 68000 waits for *REQUEST* to go low - in other words, back to step 7).

That process continues until the entire command is sent. The disk lets you know that the command is finished by releasing *C/D* or by pulling down on *I/O*. There are a number of possibilities, depending on whether this is a read, a write, or a status request, and on whether or not something has gone wrong. In any event, getting data from the disk is just like sending it to the disk, except that the *I/O* line will be low (this rigs the AS648 so that it drives the 68000 data bus when an *INOA* is executed.) The *REQUEST* at that point indicates that a byte is available. Doing an *INOA* causes that byte to be grabbed, and when complete, lets the SCSI interface know (via that same flip-flop) that the byte has been grabbed by asserting *ACK*.

The only other thing that needs mentioning is the parity generation. There are two LS280s in the circuit - the upper one generates parity when writing to the SCSI interface, and the lower one checks the parity when reading from the interface. The lower flip-flop is cleared during the select sequence, and is subsequently clocked every time *REQUEST* goes away. The little loop with the OR gate is designed to accumulate the condition of the parity checker - if ever it goes high, it stays high thereafter. The output of the lower F74 can be checked by the 68000 by doing an *INOC*.

And that's a quick introduction to SCSI bus communications. A more complete understanding can be gained only by studying the disk manufacturer's spec sheets. A more detailed step-by-step analysis of a disk read operation will be given in the diagnostic procedures section of this manual.

VIDEO TIMING SIGNALS.

Well, so much for the easy stuff. We will now start moving into the circuitry that makes the 10K/20K the special and wondrous thing that it is. From here on out, almost every thing that we will be doing will involve video signals, so let's start with the synchronization signals coming into this machine.

Please turn to sheet 5 of the schematic. For now, we are only concerned with the lower-right-hand corner. You will notice that we bring two video synchronization signals in: "Composite Sync", and "Composite Blanking". We assume that these signals are normally about zero volts and pop downward to some negative value during SYNC and BLANKING - I suppose it would have been clever to AC-couple these guys, but it didn't occur to me until just now. Oh, well. Each of these signals is brought in and converted to TTL levels by LM360N comparitors.

After conversion, these signals are given the names ***RSYNC*** and ***RBLANK***. These mean "Raw Sync" and "Raw Blanking" and are used where you really want the video timing references. Otherwise, these signals are synchronized to the internal clock of the 10K by running them through flip-flops that are clocked by **CLK**.

Besides converting composite sync and composite blanking to TTL and synchronizing them to **CLK**, this circuit has three other functions:

1) It generates a pulse at the beginning of each line.

2) It generates a pulse at the top of each field.

3) It figures out which video field you are in.

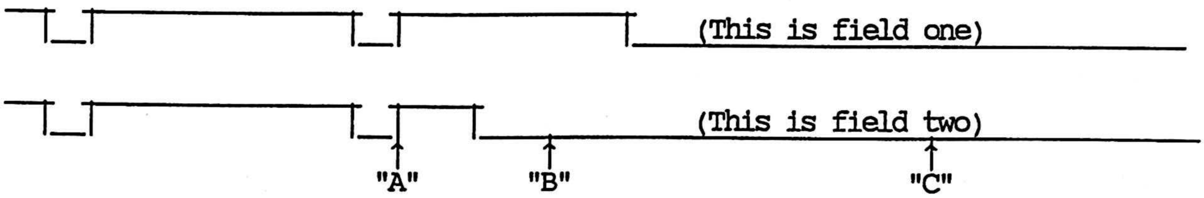
Let's examine each function.

The line pulse is the simplest - the two rightmost flip-flops in combination with the NAND gate down below generate a low-going 40-nanosecond ***LINE*** pulse every time ***BLANK*** goes from a high level to a low level. (We use blanking rather than sync in order to avoid confusion with the equalizing pulses and vertical sync pulses that occur during the vertical interval.)

The field pulse is almost as simple - the 128 microsecond one-shot is constantly being re-triggered by the rising edge of **BLANK**, which normally occurs about every 64 microseconds. When the vertical interval comes along, **BLANK** goes high and stays there for about 1.3 milliseconds. When that happens, or rather, 128 or so microseconds after that happens, the Q-not output of the one-shot will go high, which clears the first flip-flop. When **CPUCLK** comes along, the ***FIELD*** output of the second flip-flop will go low. This ***FIELD*** signal sets the first flip-flop, so when **CPUCLK** comes around again, the ***FIELD*** pulse will end. Thus, about 128 microseconds after the vertical interval starts, ***FIELD*** will go low for 80 nanoseconds.

The determination of which field you are in is a simple enough circuit, but relatively hard to figure out without a timing diagram in front of you. Although I think trying to draw timing diagrams on a typewriter is silly, I will give it a stab, since I wish to avoid the need to insert pictures and figures into the text (Oh, for a Macintosh!)

The following two lines show the difference in composite blanking at the beginning of field one and the beginning of field two:



For both fields, point "A" is where the 37 nanosecond one-shot fires for the last time (it has been firing on every rising edge of *BLANK*.

Point "B" is where the 37 nanosecond one-shot finishes. Its rising Q-not output clocks the next flip-flop in the chain. You will notice that on field one it latches a high value, and on field two it latches a low value.

Point "C" is where the Q-not output of the 128 microsecond one-shot goes high - it latches the second flip-flop in the chain. This flip-flop saves the *FIELD1* and *FIELD2* information for the duration of the entire field.

We now know what field we are in (*FIELD1* and *FIELD2*), we get a pulse at the beginning of each line (actually, *LINE* pulses downward at the falling edge of *BLANK*), and another pulse about 128 microseconds after the beginning of each field (*FIELD*). Since I am dreading describing the Dot Matrix circuitry, let's go back and look at the interrupt circuitry.

INTERRUPT GENERATION AND ACKNOWLEDGE

On the lower-right-hand side of sheet 1 of the CPU schematic you will find the interrupt generating circuit stuff. There is only one interrupt request generated (of priority level 4) but there are two different conditions which can generate that interrupt request.

The middle ALS74 is set by a *FIELD* pulse - this is the first condition. You will notice that when you get that *FIELD* pulse the interrupt-request line *IPL2* goes low. You will recall from our earlier discussion of interrupts that the CPU acknowledges an interrupt by turning FC2, FC1, and FC0 on and pulling *AS* low. This will eventually cause a 68000 memory cycle to begin, during which time *GRTCPU* will be low. A look at this circuit will you that at that time a number of things will happen:

1) The interrupt request flip-flop will be cleared.

2) The interrupt vector 040 will be placed on the data buss.

3) *DTACKOC* will be pulled low, telling the CPU to grab that vector.

4) INTA and *INTA* will be activated. These signals are used to tell all of the rest of the circuitry to lay off of the data buss.

The upshot of all that is that this circuitry will generate an interrupt request every time a *FIELD* pulse comes along. In order to do serial communications, however, the computer needs to be interrupted more frequently. We give the programmers control of the frequency by counting up some number of video lines, and then generating an interrupt request.

The selected number of lines is kept in the AS534, which is an inverting latch. Every *FIELD* pulse, or when the AS867 counter gets up to 0FF, the flip-flop at the lower left causes that number to be reloaded into the counter. When the counter gets up to 0FF, an interrupt request is generated.

Let's go over the whole interrupt requesting process again. Assume that the 68000 executed an *OUT02* and loaded a 007 into the AS534. Starting at *FIELD* time, an interrupt request will be generated. One *SYNC* pulse later, the value 0F8 (the inverse of 007) will be loaded into the AS867.

Seven *SYNC* pulses later, the counter will have gotten up to 0FF and its *RCO* output will go low. This will cause the AS867 to be loaded again with 0F8 on the eigth *SYNC* pulse, which will simultaneously cause another interrupt request to be generated. It should be clear that loading the AS534 with a value N causes interrupt requests to be generated every N+1 *SYNC* pulses.

Note that these are *SYNC* pulses, not *LINE* pulses. *LINE* does not occur for about 1.3 milliseconds during the vertical interval. This is too long a period of time if you are trying to handle communications lines at 9600 baud. Using *SYNC* means that we get too many counts during the vertical interval, but this is a case where too many is preferable to too few.

As mentioned previously, the *IN02* enable on the F244 lets the 68000 know which field is currently active, and whether or not the interrupt being serviced was caused by a field pulse, or by a line count.

I no longer have any alternative - I have to start talking about the dot matrix stuff.

DOT MATRIX CIRCUITRY

It is time for me to return, however reluctantly, to sheet 5 of the CPU schematic. In principle, the dot matrix is simple enough - every 640 nanoseconds, sixteen bits are read out of the 128K of dynamic RAM and are latched into the two F323 shift registers. These bits are then shifted out and converted to video by a 40 nanosecond clock. Since the memory cycle time is 320 nanoseconds, this leaves every other memory access available to the 68000 if it wants it. In practice, there are an annoying number of silly little things going on that have to be understood.

DOTCLK AND USDOT SYNCHRONIZATION.

The first conceptual problem that has to be addressed is the business of synchronizing the video output signal with the internal clocks of the CPU board. Item: The 68000 is running on a 12.5 megahertz clock (80 nanosecond cycle **CPUCLK**) which is derived from a 25 megahertz clock (40 nanosecond **CLK**) which is itself derived from a 100 megahertz oscillator. (Please refer to the top of sheet 2 of the CPU schematic.) All of these signals are free-running - they are not at any time interrupted or reset.

This is all very nice, but there is a problem, because none of these frequencies fits into a video frame, which as we all know represents exactly 119437.50 cycles of 3.579545 megahertz (plus or minus 0.33). In other words, even though the 10K/20K is designed to allow color changes on 40-nanosecond boundaries, that clock must be synchronized to a greater degree of precision. Otherwise, the first dots will appear with an error of plus or minus 40 nanoseconds, which is very easily visible, even in the encoded format. It is painfully obvious on an RGB monitor.

This synchronization is obtained by deriving the dot clock (which is used for both the dot matrix and the main display board) from a still faster 100 megahertz clock. This means that the video displays start with an error of 10 nanoseconds. If you put your nose right against an RGB monitor you can just barely see this - it is just not visible on an NTSC or PAL display.

Let's see how this is done. At the top left corner of sheet 5 you will see a flip-flop that is clocked to the cleared position at the beginning of blanking. That flip-flop's output goes to the input of the next flip-flop, which is clocked by USDOT. **USDOT** is the video-synchronized 40-nanosecond clock - on the DISPLAY board it is called **DOTCLK**. It's the same signal, we just gave it different names on the two boards.

Anyway, as soon as the video sync signal comes in, that second flip-flop will be release so that ***KILDOT*** goes low - note that it goes low on an **USDOT** boundary. ***KILDOT*** goes to the circuit which is diagrammed at the top of sheet 2. As you might guess from its name, its job is to kill the dot clocks - it does this by holding in the cleared condition the two flip-flops which divide 100 megahertz by 4 to get the 40-nanosecond **DOTCLK** and **USDOT**.

Back on sheet 5, I again ask you to notice that ***KILDOT*** is designed to make **USDOT** go away cleanly - **RSYNC** goes high, which lets ***KILDOT*** go low when **USDOT** high. ***KILDOT*** will force **USDOT** to go low, but the delays through the two flip-flops are such that it goes away cleanly.

On the other hand, when **RSYNC** goes low again, at the end of the sync pulse, *KILDOT* will go high again. This frees up the divide-by-four flip-flops back on sheet 2. They will start their counting sometime in the 10 nanoseconds after the sync pulse goes away.

To sum up the **USDOT** synchronization process - **USDOT** and **DOTCLK** go away in a clean fashion during the very first (and only the very first) low-going sync pulse that occurs after blanking begins. **USDOT** and **DOTCLK** start up cleanly after the sync pulse goes away - the divide-by-four starts counting within 10 nanoseconds of the end of sync.

DOT MATRIX MEMORY TIMING.

The dot matrix memory timing is similar to main memory timing - don't confuse the signal names. The names are similar, but they have nothing to do with each other - the dot matrix memory is synchronized with the video timing signals since it runs on **USDOT**. The main memory runs off of **CLK**, which is not the same - **CLK** and USDOT can be phased relative to each other in eight different ways. Let's look at the dot matrix memory timing.

First, let's assume that we are in the middle of a line - the display hardware is making use of every other cycle, and the 68000 can get into the other cycles if it wants to. The F161 counter is free-running merrily away and is basically being used as a divide-by-sixteen counter. The signal ***68CYD*** is low for eight **USDOT** cycles, indicating that the 320-nanosecond memory cycle is available for the 68000. During the other eight **USDOT** cycles ***68CYD*** is high, indicating that this is a "display-the-dots" cycle.

The divide-by-four output of the F161 feeds the input of the F379, which is set up to be a four-bit shift register. Consequently, the F379 will fill up first with zeroes, then with ones, and repeat _ad nauseum_. DDE0 is defined as that stage where the F379 is filled with ones. DDE7 is defined as the stage right before then - the F379's outputs are 1110. The dot matrix ***RAS*** starts with DDE1 and runs through DDE5; ***CAS*** runs from DDE2 through DDE6.

Having learned that, we should now go back and look at the circuit which synchronizes the dot matrix memory timing to the video. At the upper-left corner of sheet 5 we see that the very first *DDE7* pulse that occurs after the start of video blanking will cause flip-flop to be cleared. Once that flip-flip is cleared, the **USDOT** clocks that occur after blanking but before sync will cause the F161 to be cleared and the F379 to fill up with ones - this, of course is the DDE0 condition.

The circuit is held in this DDE0 stage until the leading edge of sync comes along - there's a catch, though. Because of the one-CLK delay of the **SYNC** signal, the F74 holding the F161 and the F379 in the DDE0 condition does not get clocked until **USDOT** has gone away for the **KILDOT** period. So, those two counters will stay in the DDE0 state until **USDOT** starts up again. At that point, of course, the F161 and the F379 will be properly and predictably locked up to **USDOT**.

HOLDOFF CIRCUITRY.

Well, we now have clock signals (**USDOT** and **DOTCLK** which are synchronized to the video, since they start up based on the trailing edge of video sync. That means that those clocks start up something like 4.7 microseconds before the end of video blanking. We want to hold off generating any video stuff until just before the end of blanking, so we need a *HOLDOFF* signal.

I refer you back to sheet 1 of the CPU schematic. In the middle of the right side you will find a block labeled "Holdoff Counter". Its job is pretty darned simple. When *RSYNC* comes along it sets the two flip-flops. Later on it goes away, leaving the flip-flop set for a moment. The first rising edge of **USDOT** will cause the flip-flop to clear; at the same time the AS867 counter will be loaded with the value at its parallel inputs. Subsequent **USDOT** clocks cause the AS687 to count down. When it goes past zero, the high-order output will go high. That will cause the rightmost flip-flop to be cleared.

When the rightmost flip-flop was set, it activated the *HOLDOFF* signal. When it is cleared by the counter running down, the *HOLDOFF* signal goes away. Its disappearance signals both the dot matrix circuit on this board and the main display circuitry on the DISPLAY board that it is now time to start generating video. *HOLDOFF* goes away about 0.4 microseconds before the start of active video.

Let's get back to sheet 5 and see how dots are actually generated. We will start at the top and work down.

At the very top of the sheet you will see a pair of flip-flops that have one and only one job: The very first ***DDE6*** that occurs after a top-of-file pulse will cause the upper AS867 to be cleared. You will notice that the two flip-flops are rigged up so that the clear-the-AS867 pulse goes away on the next ***DDE6*** pulse after that. This AS867 has the job of determining which video line you are on - after it has been cleared, it counts up one every time you get a ***LINE*** pulse.

The reason that we make sure that the counter gets cleared at DDE6 time is because we have no control over when a field pulse arrives. If we used it to clear the counter directly, then the counter could clear at a time when the address to the RAM chips was supposed to be stable. We have discovered that dynamic memories are prone to forgetfulness if you change their addresses at inappropriate times; hence the DDE6 business.

The second AS867 has the job of counting up which 16-bit word is going to be sent out as video. You will notice that it is held clear by our old friend ***HOLDOFF***, and once ***HOLDOFf*** goes away the counter is clocked by **DDLOAD**. **DDLOAD** is generated in the lower-left-hand corner of the drawing. It is a combination of **68CYD** and ***DDE5***. This means that ***DDLOAD*** occurs during dot access cycles at a time when the output data from the memory chips is valid.

The output of these two counters is combined with ***FIELD2*** and goes to the upper two F244s, which multiplex the address bits out to the RAMs. We'll look at that in a moment; first let's examine the addressing.

The lowest seven bits of the counters specify which 16-bit word you are on along a line. Two to the seventh is 128, and there are 16 dots per word, so these seven bits can specify 2,048 dots along a line. We are sending dots out at 25 megahertz. Since active video is about 53 microseconds, we use about 1,325 of those dots on each line. The remaining 700 or so dots are skipped.

The eighth bit of the address comes not from the counters, but rather from the ***FIELD2*** signal. This is how we handle the interlace business. Think of it this way - that bit is actually the lowest-order bit of the line counter. Thus, during field one we will be displaying lines 1, 3, 5, etc. During field two we will display lines 0, 2, 4, and so on.

Wait! I hear you cry. Line 0 in field 2 and line 1 in field 1??? What's going on here?

It's true. If you look really close at the timing diagram for NTSC video, you will discover that the very first visible line (which is actually one-half of a video line) is line 20 of field tow. I mention this now because it will be a matter of some importance when we try to figure out the line pointer table stuff on the DISPLAY board - but for now that's borrowing trouble. Anyway, here in the dot matrix, we display the even lines during field two, and the odd lines during field one.

So, now we have a valid sixteen-bit address going to the upper F244s. These two parts, which multiplex that address out to the RAMs, are enabled by the F138. You will notice that the upper two F244s are enabled when *68CYD* is high. The lower two F244s, which are in charge of sending the 68000 address out to the RAMs, are enabled when *68CYD* is low.

A quick word about refreshing. The dynamic RAMs need no special refresh cycles, since the dot display does enough accessing to take care of the refresh problem. As the note in the box declares the lines are scrambled a bit. Bits 7 and 10 are swapped so as to insure that the entire 128 count refresh space gets hit every few lines.

Back to the address multiplexing. The swap from the row addresses to the column addresses is started by the **DDRAS** signal. It is delayed though a couple of ALS08s which go into the ALS138 and causes a switch from their row address driver to the column address driver.

Go back down to the bottom of the sheet. On the left you see a pair of F323s. These eight-bit shift registers are loaded up with the data from the dynamic RAMs each time **DDLOAD** goes high. As previously explained, **DDLOAD** goes high every sixteen **USDOT** cycles at a time when the data from the RAMs is valid. Once loaded, the sixteen bits are shifted out and are immediately gated with a signal from a flip flop. Why?

Well, why not? Actually that flip-flop is there to handle the PAL problem. Not the programmable parts, I refer here to that video standard used by our friends across the ocean. The PAL video standard has 625 lines. The 128K of memory that we have here can handle only 512 lines of video - that's 256 lines on each field. The flip-flop gets set at the top of each field, allowing the dots to go through. When you get to line 256, the flip flop is cleared, forcing any remaining lines to be black.

Once though that gate, the dots are gated with RBLANK, which forces the video to blank level during video blanking. The additional lowering of the signal which is required by video sync is accomplished with the other 75451. The generation of the three possible levels - white, black, and sync (the fourth possibility - sync and no blanking - is not possible) using the open-collector drivers combined with the resistor networks is an interesting exercise - I recommend that the reader work it out.

You will note that there is no set-up level - we decided that a pure monochrome high-contrast display like this one did not need it.

68000 ACCESS TO THE DOT MATRIX MEMORY.

Once we explain how the 68000 gets at the dot matrix display memory, we will be done with the circuit description of the CPU board. Let's trace it through step by step.

The 68000 wants to do a read or write of the dot matrix memory. It sends out the appropriate address and sets the address strobe. This will cause the *DISPADD* signal to go low. Some time later the 68000 will be given a memory cycle; this will cause *CPUCY* to go low. Once this happens the the output of the F02 (left side, in the middle) will go high. This clears the flip-flop, which lets the rest of the circuitry know that a 68000 request into the dot memory is active.

The next flip-flop in the chain is the "The-68000-Is-Doing-An-Access" flip-flop. It gets clocked during *DDE0* and *68CYD*. If an 68000 dot matrix request is pending, this flip-flop's output will go low and stay low until *DDE7*. Note that when this flip-flop goes low a number of things happen.

1) It removes the "68000 dot matrix request pending" signal from the first flip-flop.

2) The low-going output of the flip-flop goes to the F10. This will, if you follow through the chain, cause *DDRAS* to go low during this cycle.

   (*DDRAS* goes low during every dot-matrix cycle. It only goes low during a 68000 access (when *68CYD* is low) when the 68000 has actually requested access to the dot matrix memory.)

   Because *DDRAS* goes low, an actual memory access is done. You will notice that during a *68CYD* cycle the 68000 address buss is sent out the the RAMs.

3a) At DDE5 time, during a read, the two AS574s will latch up the sixteen bits of data coming from the RAMs.

3b) During a write, the upper and lower byte write lines *DDW0* and *DDW1* will be driven, as specified by *UDS* and *LDS*.

4) At *DDE6* time, the third flip-flop will get clocked. This causes a *DTACKOC* to be generated, and also causes the data buss to be driven with the data read from the RAMs.

When the data strobe signal DS goes away this flip-flop will be reset - at this point the 68000 access of the dot matrix memory is complete.

It is important to understand that the clock controlling the memory accesses to the dot matrix RAM goes away for about 4.7 microseconds every horizontal interval. It also goes away for about 2.4 microseconds during the first equalizing pulse after the *FIELD* pulse. As a result, it is possible for the 68000 to do an access of the dot matrix memory and not get a *DTACK* for about 5 microseconds. If you are not aware of this, you might come to the conclusion that something is wrong if you have some reason to scope out those signals.

Believe it or not, this concludes our discussion of the circuit theory of the 10K/20K CPU board.

OVERVIEW.

I see that you are still with me. At least I hope so. Presumably you have read through the circuit theory description of the CPU board - if you have not, this section may not make a whole lot of sense.

In spite of the fact the the DISPLAY board has almost twice as many chips as the CPU board, it is actually a simpler board. Unlike the multi-functioned CPU, this board really only has two jobs:

1) It has to behave like a four megabyte memory for the 68000.

2) It must take certain information in that memory and convert it to video.

Granted, by the time we are done, that description will seem over-simplified. But keep it in mind as we discuss the board - it may keep you from getting too discouraged.

We will begin this discussion by ignoring the display functions of the DISPLAY board - we will initially examine it purely in terms of its communications with the CPU board. Let's start by looking at the interboard connections.

CPU BOARD TO DISPLAY BOARD CONNECTIONS.

Please turn to sheet 2 of the DISPLAY schematic. Over on the right side, in the middle, you will find the symbols and signals for P1, P2 and P3. These three ribbon connectors are used to bring signals from the CPU board to the DISPLAY board. Each signal is used as follows:

Connector P1:

> This connector is mostly 68000 address buss. The two additional signals are ***UDS*** and ***LDS***, which are the upper and lower data strobes from the 68000.

Connector P2:

> This connector is mostly 68000 data buss. The other signals:
>
> ***RAS***. The Row Address Strobe for the main RAM.
>
> ***CAS***. The Column Address Strobe for the main RAM.

***GRTREF*.** Indicates that this is a Grant Refresh memory cycle.

***WR*.** The memory Write line from the 68000.

***RAMEN*.** Address decode from the 68000, indicating that the 68000 is trying to get at main RAM.

***XDE345*.** This is the buffered signal which goes low during the DE3, DE4, and DE5 periods of a memory cycle. It is useful because DE5 ends 200 nanoseconds after *RAS* goes low. Since we are using 150 nanosecond RAMs on this board, *DE345* is a useful signal for driving the data buss during a read.

XGRTCPU. The buffered version of the signal that announces that this is a memory cycle devoted to the 68000.

XGRTTOP. The buffered version of the signal that this memory cycle has been assigned to the display circuitry on the DISPLAY board.

Connector P3:

XDE3*.** Buffered ***DE3.

***FIELD2*.** This comes from the CPU board video circuitry. It indicates, when low, that we are in the video field that starts with one-half of a visible line.

***OUT04*.** This output strobe is used to initialize the line pointer circuitry.

***OUT06*.**
***OUT08*.** These two output strobes latch up a 32-bit number which is used to define the start of a video image.

***PALADD*.** This is a 68000 memory decode indicating that the 68000 is writing into the palette.

XCPUCLK. A buffered 12.5 megahertz clock.

XDOTCLK. This buffered 25 megahertz clock is supremely important to the video generation circuitry on this board. As detailed in the circuit theory description of the CPU board, it goes away during video sync, and when it starts up again it is lined up with the end of video sync. This signal is used to define the 40 nanosecond boundaries of the color changes coming off of this board.

XDE456*.** This signal is very similar to ***XDE345. Its main feature is that it occurs 40 nanoseconds later.

***BLANK*.** This is video blanking.

***REQTOP*.** This signal is unique in that it is the only signal that comes from the DISPLAY board and goes to the CPU board. It signals that the video display circuitry requires access to the memory.

***FIELD*.** This signal pulses low once at the beginning of each video field.

***OUTOE*.** This 68000 output strobe is used to switch palettes.

***DSRD*.** This signal is a combination of 68000 data strobe and write signals. It goes low when the 68000 is expecting somebody to be driving its data buss on a memory read.

***SYNC*.** This is video sync.

***HOLDOFF*.** This signal goes low during horizontal sync and goes high just before the start of active video.

As you may have gathered, the signals whose names start with "X" simply mean a version of the signal which we have found needs to be buffered. On sheet 2, you will see where, for instance, **XCUPCLK** is received and changed into **CPUCLK.** You will also notice that we used two drivers where we might have used one. We did this because we noticed that there was a good deal of crosstalk within the receiver when the clocks and the slower signals were all brought into the same part.

The other thing that it is important to get straight from the beginning is that even though the CPU board and the DISPLAY board both have busses which are named **A<23:01>** and **D<15:00>** they are not the same busses. They are separated by the five chips found in the lower-right-hand corner of sheet 2. You will notice that the address buss is driven from the CPU card only during a ***GRTCPU*.** The data buss is driven only during ***RAMEN***, ***GRTCPU*** and ***WR***, and is received (that is, transferred from the DISPLAY board to the CPU board) only during ***RAMEN*** and ***DSRD*.** We will cover this again - I just wanted to mention from the outset that those busses are separated.

Let's start off by looking at a 68000 write access to the main memory.

68000 MAIN MEMORY WRITE.

There is another thing that we must get clear before going on to talk about the specifics of a memory access. This circuit board contains four megabytes of memory, since there is room for 128 256K RAMs. These are organized as a single 128-bit work - all memory chips are enabled simultaneously. The reason for this will become clear later on when we talk about the display circuitry.

The 68000, however, only does sixteen-bit accesses. The interface on writing is made by duplicating the 68000 data eight times and hitting the Write Enable lines of only the sixteen chips that are supposed to be changed - the other 112 chips do reads, which are ignored. On a read, all 128 chips are hit with the same address - only the sixteen bits we are interested in get multiplexed onto the sixteen bits of the D<15:00> buss.

We are now ready to go through a 68000 write cycle. Let's examine it on a clock-by-clock basis:

DE0: ***GRTCPU*** goes low, causing the addresses from the CPU board to be driven onto **A<23:01>**. This is a write, so ***WR*** is low. In the lower-right-hand corner of sheet 2 you will see that this will cause ***DBDRIVE*** to go low, which means that the data buss from the CPU board will be driven onto **D<15:00>**. On the bottom of sheet 2, in the middle, you will see where that data buss goes through some series resistors on its way out to the RAMs. If you look at sheets 3 and 4, you will see how that data gets repeated eight times

DE1: ***RAS*** goes low. Towards the upper-right corner of sheet 2 you will see that ***RAS*** is repeated eight times and sent out through series resistors to sixteen RAMs each. This causes the row addresses to be latched up. The ***RASA*** signal also goes down to the ALS08 in the lower-left-hand corner of sheet 2. After being delayed and inverted it goes up to the drivers which are multiplexing the eighteen-bit address down to the nine lines which go to the dynamic RAMs. It is this signal which causes those drivers to switch from the row address to the column address.

DE2: ***CAS*** goes low. By this time the address drivers have switched from the row address (while ***ENRALO*** is low) to the column address (while ***ENRAHI*** is low.

DE3: Nothing special happens here.

DE4: It is during DE4 that the write enable lines of the appropriate RAMs are pulled low.

If you look at the bottom of sheet 2, just to the left of the middle, you will see where the sixteen write lines, ***WB0*** through ***WBE*** are generated. Their names reflect their associated addresses: A byte write to location 012345B will cause ***WBB*** to go low.

Tracing back the logic is straightforward: ***UDS*** and ***LDS*** are used to determine which byte is being written; address lines 03, 02, and 01 are used to determine which of the eight words is being addressed, ***RAMEN*** and ***DBDRIVE*** make sure that the write lines are enabled only during a 68000 access of the main ram space, and ***DE456*** makes sure the enable occurs only during the proper part of the memory cycle.

DE5: A continuation of ***DE4***. At the end of ***DE5*** the ***RAS*** signal goes away. Note that the RAM address lines will change here, since when ***RAS*** goes away ***ENRALO*** will go low again.

DE6: At the end of ***DE6*** the write enables and ***CAS*** will go away.

At this point the cycle is to all intents and purposes over. Back on the CPU card another access request may be pending - it does not matter here - *RAS* won't go low again until the next *DE1*. This means that all requirements with regards to RAS and CAS precharge time, total cycle times, and whatnot will be satisfied.

You may have a question regarding ***DTACK***. The generation of ***DTACK*** for this memory is handled back on the CPU card - it does not matter whether or not the board is even plugged in. Once a ***CPUCY*** comes along when ***RAMEN*** is low a ***DTACK*** will be generated at ***DE3***.

So, we are done with the 68000 write. Let's compare it to a read.

68000 MAIN MEMORY READ.

The overall timing of a read is exactly the same as for a write. The difference is that the write enable lines do not get decoded - rather, the proper sixteen bits out of the 128 get placed onto data buss. This is done by the sixteen AS573 transparent latches which are found in the middle of sheet 2.

The F138 is in charge of deciding which sixteen-bit word is desired. Whenever *DBDRIVE* is high (which is all the time, except when the 68000 is writing) this F138 decodes address bits 03, 02, and 01 and activates one of the Read Word signals *RWD0* through *RWD7. This causes one pair of the AS573s to drive the sixteen-bit data buss.

During an actual memory cycle, the AS573s are made transparent during ***DE345***. We are using 150-nanosecond RAMs on this board - *DE345* goes low starting at 80 nanoseconds and ending at 200 nanoseconds into the cycle. Thus, when ***DE345*** goes high (causing **68RLTCH** to go low) the AS573s will hold the data that was just read from the RAMs.

When the read access has been assigned to the 68000 (meaning ***GRTCPU*** is low) the value that the selected pair of AS573s is putting onto **D<15:00>** will be driven though the two 29863s onto the data lines of connector P2.

And that is a read cycle. A word or two about refreshing, and we will be done with this sheet of the drawing.

MAIN MEMORY REFRESH.

About every ten microseconds or so, the CPU will send a *GRTREF* over to this board. If you look at the upper-right-hand corner of sheet 2 you will find an eight-bit counter and a driver which places the output of that counter onto the even-numbered bits of the address buss.

It's really quite simple. Every ten microseconds *GRTREF* goes low. The counter gets put onto those address bits (which are the ones which are enabled during *ENRALO* time) and a read is done of that location, which refreshes it. At the end of *GRTREF* the counter gets clocked, and everything is set up for next time.

We can now start looking at the hard stuff.

GENERAL DESCRIPTION OF THE VIDEO DISPLAY CIRCUIT.

Let's go through the whole chain of logic which ends up with video signals being generated. Once we know what is supposed to be happening, we can then go back an figure out how it is being done.

1) A picture starts with the 32-bit SHELL POINTER. This value (which lives in a hardware register) points to the beginning of the LINE POINTER TABLE.

2) The LINE POINTER TABLE lives in memory. It is a list of 32-bit values, one value for every video line. Each value points to that line's LINE DATA.

3) The LINE DATA for a line consists of a stream of sixteen-bit values. These values are the run-length-encoded data for the video line. The meaning of the various bits:

a) Bits 06-00:	This is the length of the segment, and can be any value from 2 to 127. A value of 1 or 0 will cause the line to terminate at that point.
b) Bit 07:	This is the cursor bit - setting this bit will cause the cursor signal to overlay the video signal on the edit display.
c) Bits 15-08:	This is the eight-bit color index. It points into the 256 entry COLOR PALETTE.

4) The COLOR PALETTE holds 256 colors, each of which is specified with 32 bits. There are eight bits each of red, green, and blue, so there are over 16 million colors possible. The other eight bits are used to specify an eight-

bit soft key value.

Each count of run-length represents 40 nanoseconds. As I mentioned above, the minimum run-length is two counts, or 80 nanoseconds. We can read out eight segments at a time, since our memory is 128 bits wide. Suppose all eight segments are of the minimum two-dot length. Eight two-dot segments, at 80 nanoseconds each, represents 640 nanoseconds. Since our memory cycle time is 320 nanoseconds, this means that we can display any number of two-dot segments, simply by doing display accesses every other memory cycle. Perhaps now you appreciate why we made our memory 128 bits wide. If it were any narrower, we would not be able to keep up with extended runs of two-dot segments.

FIELD/LINE VIDEO TIMING SIGNALS.

Before diving into the actual guts of the circuitry, lets first talk about a couple of extra video timing signals that this board uses.

The first set of signals are developed at the middle of the top of sheet 1 of the DISPLAY schematic. They are derived from the ***FIELD*** pulse that occurs at the top of each video field. The first signal is ***FIELD+0+1*** and is a combination of the first and second low-going delayed versions of the ***FIELD*** pulse. The shift register also generates the additional delayed pulses ***FIELD+3***, ***FIELD+4***, and ***FIELD+5***.

Okay, so those guys are a stream of pulses that occur once every video field. We also have a set that are derived from ***SYNC***, so they occur every video line - more frequently during the vertical interval. These are called ***LINEC1*** through ***LINEC6*** and are generated by the circuitry just below the ***FIELD*** stuff.

QUICK INTRODUCTION TO THE AMD 2940 ADDRESS GENERATORS.

A little more background material, and we can let loose on the circuitry. The addressing into the LINE POINTER TABLE and the addressing into the LINE DATA is done with some Advanced Micro Devices 2940 DMA address generators. These are fairly clumsy, and fairly slow, and use a little too much power, but I can use six of them instead of twelve other parts. I recommend that you look them up in an AMD data book, but basically they are each eight-bit counters with 48-milliampere tri-state buffers on their outputs.

They each have eight different instructions that they can execute - of that eight we use three. These are:

1) Load Command Register - we use an ***OUT04*** to tell the 2940s that when we issue the 'count' instruction later, we mean a simple 'count up'.

2) Load Address Register - When this instruction is executed, we load the 2940s with whatever value we wish to count up from.

3) Count - When this instruction is executed, the address value inside the 2940s is incremented.

The 2940s are arranged in two groups of three. The top set of three is labeled "Line Pointer Addresses". This group is reset to the SHELL POINTER value at the top of each field. During field two, it is bumped ahead one line to account for interlacing. On every set of ***LINEC*** pulses, it gets bumped ahead two lines worth. In other words, this set of address generators is in charge of keeping track of where to find the next entry in the LINE POINTER TABLE.

The bottom set of 2940s gets loaded with a new address at the beginning of each line. Every eight segments it gets bumped by one count. This set of 2940s is in charge of maintaining the address of the next set of segments to read from memory when working along a line.

OVERVIEW OF THE ADDRESS GENERATION SEQUENCE.

1) At the very top of each field, the upper set of 2940s is reloaded and points to the very first entry of the LINE POINTER TABLE.

2) If this is field two, the upper set of 2940s is bumped so that it points to the second entry of the LINE POINTER TABLE. This is to handle the fact that on alternate fields you display alternate sets of lines.

3) At the very beginning of each video line (right after ***SYNC*** goes low) the upper set of 2940s is bumped twice. This accounts for the fact that on each field you display one-half the lines.

4) After the upper set of 2940s has been updated at the beginning of each line, we are ready to load up the lower set of 2940s with address of the beginning of the run-length data for that line.

For each new line:

a) A memory request is created by pulling *REQTOP* low.

b) When *TOPGRT* comes along, the upper 2940s are enabled, which puts the address of the next LINE POINTER TABLE entry onto the address bus.

c) When *DE345* comes around, the value read out of memory at that address is loaded into the lower set of 2940s. This is the starting location of the run-length data for that line.

This description is somewhat oversimplified. I did that on purpose. It is critical that you understand this process in the simplified form. If you don't, it will be completely hopeless when we investigate it in more detail.

Okay, are you ready? Let's start with some of that detail.

LINE POINTER ADDRESS CIRCUITRY.

The first thing that we have to do is set the Command Register inside the upper 2940s to zero. We do this with the following sequence of instructions:

```
MOVEW  $00,0FFFF06  ;This clears the high-order SHELL POINTER
MOVEW  $00,0FFFF08  ;This clears the  low-order SHELL POINTER
MOVEW  $00,0FFFF04  ;This loads the upper 2940s with the SHELL
                    ;  POINTER, which is zero.
```

You will have noticed that the 2940s are controlled by PAL chips. The part labeled "PAL1" is a 16R4 - its logic, along with that of the other PALs used in the 10K, can be found in the Appendix. This particular PAL has but three major functions:

1) To generate a Load Command Register instruction when an *IN04* comes in.

2) To generate a Load Address Register instruction when a *FIELD+0+1* comes in.

3) To generate a Count Address instruction under the following conditions:

   a) During field two and *FIELD+3*.

   b) During field two and *FIELD+5*.

   c) During any of *FIELDC1*, *FIELDC3*, *FIELDC3*, or *FIELDC4*.

You should notice at this time that in order to move up one entry in the line pointer table, it is necessary to hit the counter twice. This is because the addresses point to sixteen-bit values in memory - each LINE POINTER TABLE entry is a 32-bit number, so it is necessary to bang it twice.

The PAL is also in charge of making sure that 2940s's instruction and data setup and hold times are met - these times are somewhat lengthy, which accounts for the rather circuituous logic that is found inside that PAL.

Although I have not said so specifically, it should be clear by now that the SHELL POINTER, which defines where the entire picture is in memory, is contained in the three AS574 latches in the upper-left-hand corner of sheet 1. It is this 23-bit number which gets loaded into the upper 2940s at the very top of each field.

DATA POINTER ADDRESS CIRCUITRY.

We have seen how the upper set of 2940s gets loaded at the top of each field, bumped one memory line at the top of field two, and bumped by two memory lines at the beginning of each video line. It is now time to see just how the activities of the lower set of 2940s are controlled.

It might, however, be helpful to look ahead a little and find out a little about how the LINE DATA is handled once we start reading it out of memory.

Over on the right side of sheet 1 you will see sixteen AMD 29521s. You had better read up on these parts. They are four-level pipeline registers; we are using only two of those levels. We are using them as double-buffers - the current set of eight segments is held in register A2 while the next set of eight segments is waiting in register A1. When the current set of eight segments is depleted, an instruction is given to all of the 29521s which loads up register A2 from register A1.

At the same time that happens a memory request is generated. When the grant comes around, the 128 bits of data from the RAMs are loaded up into the A1 register using the instruction which loads A1 without affecting A2. This continues thoughout the line.

Obviously, at the beginning of a line, there has to be some complicated activity which loads up all of those 29521s with the first two sets of eight segments. Let's take a look at what happens at the beginning of a line.

The beginning-of-a-line logic is handled in the top-middle of the sheet by the LS161 and the LS138 which generate the signals ***TLDHI***, ***TLDLO***, ***TLD1ST***, ***TLD2ND*** and ***TLINE***. You will notice from that part of the schematic that the LS161 gets reset once each line by the ***LINEC4*** pulse. This causes the ***TLDHI*** pulse to go low. You will also see that this 'state counter' gets advanced every time we get a ***GRTTOP*** and a ***DE456*** pulse. The last thing to note is that once this little state counter gets into the ***TLINE*** state, the LS161 is disabled, so it will stay there until the next ***LINEC4*** comes along and resets it.

The definitions of these various states are as follows:

TLDHI It is during this state that the upper 2940s are output-enabled and the address of the high-order half of the current LINE POINTER TABLE entry is placed on the address buss. At the end of ***DE345***, the high-order LINE POINTER TABLE entry will be loaded into the lower set of 2940s.

TLDLO It is during this state that the same thing occurs for the low-order half of the current LINE POINTER TABLE entry. The only difference in the address will be in address bit 01. (It is a requirement that LINE POINTER TABLE entries start on 32-bit boundaries in memory.) This bit, at this time, is controlled by the PAL2 down by the lower 2940s.

TLD1ST During this state the lower 2940s are output-enabled. This places the address of the first eight segments onto the address buss. (Notice how these 2940's don't bother with address bits 03, 02, or 01. These 2940s are only concerned with addressing sixteen bytes at a time.) During ***DE345*** the 128 bits from the RAMs get loaded into register A1 of the 29521s. It is at this time that the lower 2940s get clocked so that next time they point to the next group of eight segments.

TLD2ND During this state, the lower 2940s are again output-enabled. When ***DE345*** comes around the 128-bit data will be loaded into register A1 of the 29521s, while simultaneously the A1 data will be latched into register A2. At the same time, the lower set of 2940s will again be clocked, setting up the address for the next group of eight segments.

TLINE When this signal goes low, it means that you are now in a normal line mode. From here on out, whenever the A2 register is exhausted the A1 register will be transferred to the A2 register, a new memory request will be generated and the resulting data loaded into the A1 register, and the lower set of 2940s will be clocked.

As usual, I have left something out, because I was concerned about being too confusing. I have been talking about "exhausting" the A2 register in the 29521s, but I have not defined what that means.

Let's back up one moment. We have learned what happens during TLDHI, TLDLO, TLD1ST, and TLD2ND. At the end of all that the first group of eight segments that are going to be displayed is in 29521 register A2. Let's look at that more closely.

If you look at the 29521s, you will see that their data input pins are connected to the 128 bits of output from the dynamic RAMs. The 29521 data output pins are combined together into a sixteen-bit bus called the **TD<15:00>** buss. Which pair of 29521s drives the **TD** buss is controlled by a pair of S195s in the middle of the sheet whose outputs are called *TE0* through *TE7*.

If you look at those S195s and the LS138 which initializes them you will notice that the S195's are initialized by a 3-line to 8-line decode of three signals that we have not talked about yet: **TA3**, **TA2**, and **TA1**. These signals come from the ALS574 down in the lower-left-hand corner of sheet 1 - it is time for me to explain this part. You will have also noted that I have not made any mention of PAL4 which sits in the middle of this drawing like a spider in its web.

Again, I have not made any mention of the **TA** signals or of that PAL4 because I did not think that it was time. I think that now, however, you are ready to withstand the full fury of the beginning-of-the-line logic. Let's go through it, step by step.

1) The line starts when we get a low-going *SYNC* pulse. This causes the string of *LINEC* pulses which cause the upper 2940s to update their pointer to the next entry in the LINE POINTER TABLE.

2) *LINEC4* causes *TLDHI* to go low.

3) *LINEC6* causes the F74 in the middle of the sheet to clock, which brings *REQTOP* low for the first time.

4) *GRTTOP* goes low for the first time. *TLDHI* is still low, so the upper set of 2940s are output-enabled. PAL2 clears address bit 01 so that the high-order half of the LINE POINTER TABLE entry is addressed.

5) *DE345* comes around. This causes the topmost of the lower set of 2940s to be loaded with bits 10-04 of the high-order half of the LINE POINTER TABLE ENTRY. Bits 03-00 get latched into the ALS574 in the lower-left-hand corner as bits **TA3** through **TA0**.

   At the same time, PAL4 causes *REQTOP* to go low again.

6) When *DE456* goes down and up again, *TLDHI* goes away and is replaced with *TLDLO*.

Phew! Let's rest for a moment. It's hard to believe that all that took only about 0.5 microseconds. Ready? Let's go.

7) *TOPGRT* is still low (and it will stay low, as long as we keep pulling down on *REQTOP* - this access is, you will remember, the highest priority) and now *TLDLO* is low, so PAL2 sets address bit 01 to point to the low-order half of the LINE POINTER TABLE entry.

8) This time, when *DE345* rolls around, PAL2 will clock the other two 2940s. This will load **TA3** through **TA0** along with data bits 15 through 04 of the LINE POINTER TABLE entry into those parts. It will also clock the ALS574 again, so now **TA3** through ***TA0** will reflect the bottom four bits of the LINE POINTER TABLE entry.

9) As during the *TLDHI* cycle above, *REQTOP* will be yanked low again, and the state counter will advance from *TLDHI* to *TLD1ST*.

Time to take a breather again. What have we got? The lower set of 2940s are now ready to address the first group of eight segments for this line. **TA3** through **TA1** contain a number which indicates exactly which one of those eight segments is really the very first one to go out. (If it were not for this information, every single line would have to start on a 128-bit boundary in memory.) *REQTOP* and *TLD1ST* are both low, so we are ready to keep doing. Let's do it.

10) Because *GRTTOP* is low, but neither *TLDHI* or *TLDLO* are low, the lower set of 2940s are output enabled. This puts the address of the first group of eight segments onto the address buss.

11) When *DE345* comes around, a number of things happen:

    a) PAL4 clocks the 128 bits of data from the RAM into register A1 of the 29521s.

    b) PAL4 also does a parallel load of the two S195s. Since the LS138 has been decoding TA3 through TA1, this parallel load establishes the initial value of *TE0* through *TE7*.

    c) The lower set of 2940s are clocked so that next time they will point to the second eight-segment group of this line.

    d) The *TOPREQ* is set yet again.

12) When *DE456* goes high we move from the *TLD1ST* to the *TLD2ND* cycle.

That brings us to the end of another cycle. At this point *TOPGRT* and *TLD2ND* are low, the lower set of 2940s are pointing to the second group of eight segments for this line, one of the eight signals *TE0* through *TE0* is low, and the first group of eight segments has been loaded into register A1 of the 29521s. We are all set to go for the second group of eight.

Hang onto your hats, because believe it or not, this upcoming cycle is the most complicated of the bunch!

12) The lower set of 2940s are putting out the address of the second group of eight segments.

13) When *DE345* comes along, the second group of eight segments will be clocked into the 29521s's A1 registers, while simultaneously the first group of eight segments are transferred from the A1 registers to the A2 registers.

    Note that because the correct initial condition of *TE0* through *TE7* has already been established, roughly 20 nanoseconds after this clock the *TD* buss will have the correct data for the very first segment of the line.

14) If you look at the input pins to the PAL4, you will notice that the *TLD2ND* input is delayed one cycle - this allows the *TD* buss to settle into the very first segment before the following occurs:

While the delayed *TLD2ND* has been low, the PAL4 has been holding three of its outputs low - the **TLOAD** signal, the **TLATCH** signal, and the clock line to the two S195s. When the delayed *TLD2ND* input goes high, all three of those signals go high. Three things happen:

a) The first TCOLOR byte gets latched into the Am29821 (bottom middle of sheet 1.)

b) The first run-length gets loaded into the two F163s. (Notice that they are not counting yet - *HOLDOFF* is still low.)

c) The *TE0* through *TE7* signals get rotated to reflect which pair of 29521s the second run-length segment will come from.

15) This time around, the PAL4 will not have pulled down the *REQTOP* line - the state counter will have moved into the *TLINE* state, and the hardware is now waiting for the segments to start clocking out normally.

RECAPITULATION OF THE START-OF-THE-LINE LOGIC.

A while ago I told you what I was going to tell you, then I told you, and now I am going to tell you what I told you:

1) During *LINEC1*, *LINEC2*, *LINEC3*, and *LINEC4*, the address in the upper set of 2940s gets updated to point to the next value in the LINE POINTER TABLE.

2) *LINEC4* resets the state counter to *TLDHI*.

3) During *TLDHI* the high-half of the value from the LINE POINTER TABLE gets loaded into the uppermost of the bottom set of 2940s; part of the value from the LINE POINTER TABLE gets stored temporarily in the AS574 at the bottom left of sheet 1.

4) During *TLDLO* the low-half of the value from the LINE POINTER TABLE, along with the four bits temporarily stored in the AS574, gets loaded into the lower two of the bottom set of 2940s. That set of 2940s now contains the address of the LINE DATA for this video line. At the same time, the AS574's outputs get replaced with the bottom four bits of that address - those four bits determine which segment in the first group of eight is actually the first to go out.

5) During *TLD1ST* the first group of eight segments is loaded into the A1 registers of the 29521s. Simultaneously the two S195s are parallel-loaded with the initial condition of the *TE0* through *TE7* state.

6) During *TLD2ND*, at *DE345* time, the first set of eight segments are transferred to the 29521 A2 registers, while simultaneously the second set of eight segments is loaded into the A1 registers. The *TD<15:00> buss will take about 20 nanoseconds to settle into the run-length data for the very first segment.

7) When the DOTCLK-delayed version of *TLD2ND* goes high, that first run-length segment will be latched into the 29821 and the two F163s at the bottom middle of the drawing. As that happens, the S195s containing *TE0* through *TE7* will be clocked, which will rotate the *TE* signals into place for the second run-length segment.

8) At this point, *TLINE* will be low, indicating that the circuit is now in the normal clock-along-a-line mode. This mode will become active once *HOLDOFF* goes high.

There! You are now an expert in how the DISPLAY board initializes the first two sets of eight segments into those 29521s. Or rather, you will be an expert once I tell you about the one exception which I have, as usual, left until the end.

THE 0xxxxxE EXCEPTION.

The sequence of events that I have just detailed works fine if the LINE POINTER TABLE entry has any address whose last hex digit is 0, 2, 4, 6, 8, A, or C. Suppose, for instance, that the last hex digit is a "2". In that case, at the end of *TLD2ND*, *TE1* will be low, the second run-length segment in the first group of eight will be TLATCHed, and the S195s will rotate so that *TE2* is low for the next time.
Or suppose that the LINE POINTER TABLE entry is a value that ends with "C". In that case, *TE6* will be low at the end of *TLD2ND*. Thus, the very first TLATCH will grab the seventh run-length segment of the first group of eight, and *TE7* will then go low. Consequently, the next TLATCH that comes along will grab the eighth run-length segment out of the first group of eight.

Now consider the case where the LINE POINTER TABLE entry is a value that ends in "E". At the end of *TLD2ND*, *TE7* will be low. The first TLATCH will grab the eighth run-length segment of the first group of eight. The S195s will be clocked, and then *TE0* will go low, which wraps around and enables the first run-length segment stored in the 29521s. Unless we do something special, when we start clocking through the run-length data, we will work our way through the first set of eight segments. This is no good - only the last one is wanted.

The PAL4 comes to the rescue. If you look at the Boolean logic for that thing, you will see two signals that have not been mentioned until now. **SPCLK** (which stands for "Special Clock") is generated during while the second group of eight segments is being loaded into the 29521s provided that ***TE7*** is low at the same time. At the same time, ***REQTOP*** will be pulled low yet one more time.

When **SPCLK** pulsed low, it clocked a flip-flop whose output comes back into PAL4 and is called **SPLOAD**. This means "Special Load" - when it is low and ***DE345*** comes around the PAL4 will simultaneously shift the second group of eight segments into the 29521 A2 registers while loading the A1 registers with the third group of eight segments from the RAMs.

Meanwhile, the first **TLATCH** will have loaded the eighth segment of the first group of eight into the 29821 and ***TE0*** will have gone low. Once the ***SPLOAD*** load of the 29521s is done, the second group of eight segments will move into the A2 registers, and the very first of those eight will appear on the **TD** buss, and will be ready for the second **TLATCH**.

There are no more exceptions - you really do know everything about what goes on at the beginning of a line.

NORMAL LINE PROCESSING.

We have just, with great labor, covered what happens at the beginning of each line. The end result is this.

1) ***TLINE*** is low, which means we are ready to do normal line activity.

2) The very first segment's color has been **TLATCH**ed into the 29821.

3) The very first sement's length has been **TLOAD**ed into the two F163s.

4) ***TE0*** through ***TE7*** (and the 29521s, if necessary) have been updated so that the **TD<15:00>** buss is ready with the second segment.

5) ***HOLDOFF*** is still low, since we have not yet reached the end of video blanking.

All of a sudden, ***HOLDOFF*** goes high. In a panic, we wonder: What do we doe now? The answer, of course, is that we go into normal run-length processing mode.

You will have noticed that I have been fibbing a little. When **TLOAD** goes low, it enables the parallel input function of the two F163s. When this happens, those two up-counters get loaded not with the run-length, but rather with the ones-complement of the run-length. I will therefore refer to the count in those counters as a negative number.

Anyway, since, ***HOLDOFF*** has just gone high, those two F163s, which are rigged up as a single eight-bit up-counter, start merrily counting **DOTCLKs**. You will notice that the F138 that follows is wired up so that when the counters reach a hex value of 0FD, then pin 10 will go low. 0FD corresponds to minus 2.

Pin 10 of the F138 is synchronized to **DOTCLK** by the F74 whose output is called ***TEOS***, which means "End-Of-Segment". The ***TEOS*** signal goes up to PAL4 where it does a number of things:

1) It always causes new **TLATCH** and **TLOAD** pulses, causing a new run-length segment to be loaded into the 29821 and the two F163s.

2) It always causes the ***TE0*** through ***TE7*** signals to be updated by clocking the S195 shift registers.

3) If the ***TEOS*** occurs while ***TE7*** is low (meaning that we have just used up all eight segments contained in the A2 registers of the 29521s) the following things happen at the same time that **TLATCH** and **TLOAD** go from low-to-high:

    a) The next of eight segments are transferred from the A1 registers to the A2 registers inside the 29521s.

    b) ***REQTOP*** is pulled low, since we need to grab another set of eight run-length segments from memory. When ***TOPGRT*** and ***DE345*** are both low, the 128 bits from the RAMs will be loaded into the A1 registers of the 29521s.

Let's look at a quick example. A segment whose run-length is four is followed by a segment whose run-length is five:

Start:	Counter is -4.	
Clock:	Counter is -3.	
Clock:	Counter is -2.	Pin 10 of the F138 goes low.
Clock:	Counter is -1.	***TEOS*** goes low. This means that on the next clock, the following run-length segment will get loaded into the 29821 and the F163s.
Clock:	Counter is -5.	
Clock:	Counter is -4.	
Clock:	Etc., etc.	

ENDING A VIDEO LINE.

In the format that we use, if a run-length segment is specified with length of zero, whatever color is specified remains true from that point to the end of active video. If you examine what happens when a length of zero gets loaded into the two F163s, you will understand how this happens:

When a length of zero gets loaded at **TLOAD** time, the outputs of the F163s go to 0FF. This causes the ***TEOL*** (End-Of-Line) signal to go low. The ***TEOL*** signal feeds back into the F163s, disabling the counting function. Thus, when you get into the ***TEOS*** state, you are stuck there forever. ("Forever" is defined as that amount of time until ***HOLDOFF*** again goes low.)

29521 CONTROL AND CLOCK SIGNALS.

Without going into too much detail, it might be a good idea to quickly go over the signals going to the 29521s. First you should notice that although they have been given unique names, the signals **TI0**, **TS1**, and **TS0** have been connected directly to +5V and GND. Second, because we discovered in the breadboard stage that the 29521s are finicky about their clock and control signals, not only did we give them their own F244 driver, but we split each signal into two (so that one F244 output only drives eight of the 29521s) but we terminated them to boot.

If you look at that driver, you will see that the clock line, in addition to being delayed by the ALS08, also does a kind of loop-the-loop through the F244 - this insures that the 29521 instruction-to-clock setup time is satisfied.

GENERAL DISCUSSION OF THE COLOR PALETTE.

We now have seen how the display circuitry gets initialized at the top of each field. We have examined how the LINE DATA stuff gets set up at the beginning of each line, and we have also learned how a new segment gets latched up when the previous segment ends. It is now time to discuss just how the eight-bit **TCOLOR** (which is **TLATCH**ed by the 29821) gets turned into a 16 million color RGB value.

Please turn to sheet 5 of the schematic. Here is where you will find the palette memory and the digital-to-analog conversion circuitry.

In the middle of the sheet you will find sixteen AMD Am93422 memory chips. These are fairly fast static RAMs. They each hold 1,024 bits, arranged as 256-by-4. They are arranged in two groups, each group contains 256-by-32 bits. If you examine the signals *THIPALET* and *TLOPALET* you will see that one group of 32 bits are output-enabled by *TLOPALET*, and the other set are activated by *THIPALET*. As you will see, those two signals are always opposites: At any given moment, one will be high and the other low. Thus, one set or another of the two 32-bit groups of static RAM is always output-enabled.

If you look just below the RAMs, to the left, you will see the circuitry that controls how the 68000 is able to write into these RAMs. Assume that *TLOPALET* is low. (This means that the eight left-side RAMs are output enabled.) Let's examine what happens when the 68000 attempts to write into the palette address space.

Because the 68000 is trying to write into the palette, the *PALADD* signal is low. Because it is a write, the *WR* signal is low. Eventually, the *GRTCPU* signal will go low, and a little while after that, so will the *DE345* signal. At that moment, the F138 will be enabled.

Because *TLOPALET* is low, that means that either pin 15 or pin 14 of the F138 will go low, depending on the state of address bit 01. Pin 15 and pin 14 drive the signals *WLOL* and *WLOH*, both of which, you will notice, go to the palette RAMs on the right. Remember that it is the palette RAMs on the left which are output-enabled.

The exact opposite is true when it is *THIPALET* that is low. In that case, it is the right-side palette RAMs which are output-enabled, and it is the left-side palette RAMs which get pulsed during a 68000 palette write.

The effect of all this is that there is a "visible" palette and an "invisible" palette. The colors that come off of the board are being determined by the "visible" palette, while any 68000 write access goes into the "invisible" palette. After the software gets the "invisible" palette set up to its satisfaction, it will cause the states of *TLOPALET* and *THIPALET* to switch. This will cause the "invisible" palette to become "visible", and vice-versa.

Before we look at how the 68000 makes that switch, lets look at where the addresses come from. Over on the left, you will see two F244s and two AS574s. These buffer either the 68000 address buss or the **TCOLOR** buss out to each of the two groups of palette RAMs, depending on *TLOPALET and *THIPALET*. If you look at it for a moment the routing should become quite clear. We will talk about how the **TCOLOR** values get latched into the AS574s in a moment.

SWITCHING THE TWO PALETTES.

Which palette is visible is controlled by a couple of flip-flops on the extreme left side of sheet 1 of the DISPLAY schematic. When the 68000 executes an ***OUTOE*** with data bit 1 turned on, the first flip-flop gets set. Once it has been set, the next **FIELD+4*** pulse (which occurs only once at the top of a new video field) will cause the second flip-flop to change state. Immediately afterword, the first flip-flop will be cleared by ***FIELD+5***.

You will notice that the two complementary outputs of that second flip-flop are the signals ***TLOPALET*** and ***THIPALET***.

GENERATING THE RED, GREEN, BLUE, AND KEY SIGNALS.

We now know about the "invisible" and "visible" palettes - we still have to see how the actual video signals get generated.

First, **TLATCH** goes down and up again. This causes a new value to appear on the eight-bit **TCOLOR** buss. If you look at the extreme left of sheet 5, you will see that **TLATCH** also goes into a shift register driven by the 40-nanosecond **DOTCLK**.

This means that 80 nanoseconds after **TLATCH** goes high, the signal **PCLK+2** goes high. This is the signal that latches **TCOLOR** into the two AS574s which drive the the address lines going to the "visible" palette RAMs. The 80 nanoseconds gives the **TCOLOR** lines plenty of time to settle.

The next output of the shift register goes into the J input of a J/K-not flip-flop (we are actually using an S195, whose input is a J/K-not flip-flop.) If you work out the timing of that little circuit, you will find that the **VCLOCK** signal always goes from low-to-high 80 nanoseconds after **PCLK+2** goes high - in other words it always goes high at time PCLK+4. The trick is that it will continue to divide **DOTCLK** by two, so after that **VCLOCK** will clock away at an 80-nanosecond rate.

We needed to do this because the AH8308T digital-to-analog converters latch up the ***SYNC*** and ***BLANK*** inputs - if they don't get a continuous clock, then then they don't generate sync and blank voltage levels properly.

So, to summarize: **TLATCH** does a low-to-high transition, which latches up new eight-bit **TCOLOR** information. Two cycles later, which gives that buss time to settle, the AS574s are latched, which changes the addresses going to the "visible" palette RAMs. Two cycles later, which gives the RAMs time to do their thing, **VCLOCK** will go up, which latches the 32 bits of data from the RAMs into the four eight-bit digital-to-analog converters.

DIGITAL-TO-ANALOG CONVERSION.

At this point, things couldn't be simpler. Four eight-bit values get sent to four eight-bit analog converters, get converted to four video-format voltages, get filtered by four 75-ohm filters, and get sent out to the world via four BNC connectors. Typically, three of those signals will get used as red, green, and blue voltages while the fourth is used as a key signal.

For normal RGBK operation with a commercial encoder these signals go out as they are. For the Dubner Computer inexpensive encoder circuit the RGB units have to be modified so that the AH8308T chips supply the proper level of setup. Some keyers (I believe that the Grass Valley Model 300 switcher with linear key capability is an example) require a sync signal on their key signal. Some users may want to use the 10K/20K in a component environment, in which case the luminance channel will require a sync signal, and the others may or may not require setup. For some component systems, it may be necessary to rework two of the channels so that they are AC-coupled, and center on zero during blanking. And so on. We have not figured out what to do in all of these possible cases - at this point we just hope that we have the hooks that we need when we need them.

CURSOR INSERTION.

One final circuit, and we are done with this board. We send out RGB values, and we expect that out in the world that somebody converts those levels to an encoded signal. We expect that the encoded signal will be brought back onto this board - it is called "VIDEO IN" - so that we can stick the cursor output on top of it. This is done at the bottom of sheet 5.

The **TCURSOR** bit is latched up by the 29821 at the same time as the **TCOLOR** bits. The cursor bit goes to a series of F164s, which delay it in order to compensate for the encoding delay through an encoder. The last stage of the delay is adjustable, so that the total delay can be varied from 13 clocks up to 24 clocks - a period ranging from 520 nanoseconds to 960 nanoseconds.

If the cursor bit is off, the "VIDEO IN" line is just sent out again as the "CURSOR OUT" signal. If the cursor bit is on, the "VIDEO IN" signal is switched out and replaced with a constant voltage which produces a fairly bright gray level.

And that, friends, is the 10K/20K DISPLAY board.

OVERVIEW.

This section is primarily written for use by the people who are turning on a CPU board for the very first time. It is sincerely hoped that this will also be of some use to service people out in the field. It is true that these checkout procedures make references to test programs and equipment that may not be generally available away from the factory; on the other hand, out in the field you will be working with a board that once upon a time worked. With some luck, these descriptions might help field service people find out what's wrong.

In any event, what we are going to do here is explain, step by step, how a CPU board is tested out the very first time it is powered up. The very last step of this process process will be to test the CPU board with a known good DISPLAY board - one that is kept in the factory and never shipped to a customer. At the end of this section you will find a checkout sheet which corresponds to the detailed description here.

If we do this properly, we will be able to build and test CPU boards and put them on a rack. We should be able to build and test DISPLAY boards the same way. It then should be possible to take any pair of those boards, put them together and end up with a working 10K/20K.

To put it another way - our goal is to build up and test each circuit board on its own. When a final system is built, and goes into final test, there should be nothing wrong with it - any mistakes, short circuits, bad chips, or whatever should have been found on the board level. If there are mistakes finding their way out to the final test level, it means that the board level procedures and tests are incomplete, and should be improved.

Enough lecturing - most of us understand the need for this kind of procedure anyway. Let's start looking at a brand-new CPU card.

Thisdescription uses the circuit designators of 4357-001-2Rev C.

SET UP THE TEST SITE.

At this point you need a non-conductive benchtop large enough to hold the CPU board. You need a power-supply with a 10K/20K power connector. Obviously, a 10K/20K frame will work quite nicely as a test jig. You will need a voltmeter, and you will need a 100 MHz oscilloscope. It should be needless to say that the test equipment ought to be properly calibrated.

You will also need a properly assembled CPU board. For the time being we will need no other board - the CPU board will be tested on its own.

TURN ON THE POWER.

Plug the board in and turn on the power. Stand back and wait a few moments - if any of the chips have been placed in their sockets upside down, you will soon know about it. Be aware that reversed tantalum capacitors take hours or days before they do their explosion routine. Before going any further, double-check the power supply voltage, both at the connector, and at a bypass capacitor at the opposite corner of the board. Adjust the power supply, if necessary. It ought to be possible to keep both voltages in the range of 4.9 to 5.1 volts. Also check and adjust the -5V, +12V, and -12V supplies.

CHECK THE CLOCK CIRCUITRY.

One of the problem areas we have seen is in the counting circuitry which brings us from the 100MHz oscillator down to the 25Mhz **CLK** and the 12.5MHz **CPUCLK**. First, just verify quickly that the **CLK** exists at U11-18 [sheet 2]. Next, verify that the countdown circuitry is functioning properly. Clip onto **68CLK** at U32-7 [sheet 1]. This signal must be an extremely stable 12.5MHz - 40 nanoseconds down, 40 nanoseconds up. Any jitter or irregularity will cause the machine to behave foolishly.

If there is a problem, it probably will be found in the flip-flops connected to the 100MHz oscillator. We have found that sometimes an S74 works better than an F74, even though S74s are not guaranteed to work at 100MHz.

CHECK OUT THE VIDEO TIMING CIRCUITRY.

Before going any further, we have to establish that the video timing circuits are working properly. First, apply composite sync and composite blanking to connectors P4 and P5 (don't get them mixed up!)

Verify that ***RSYNC*** and ***RBLANK*** are following composite sync and composite blanking. (Looking at them with two probes, simultaneously, is a good way of making sure that you have sync and blanking hooked up properly.)

Verify that ***SYNC*** follows ***RSYNC*** and that ***BLANK*** follows **RBLANK***.

Trigger on the falling edge of ***BLANK***. Verify that you get a 40-nanosecond low-going ***LINE*** pulse every time ***BLANK*** goes low.

We have to check the two one-shots:

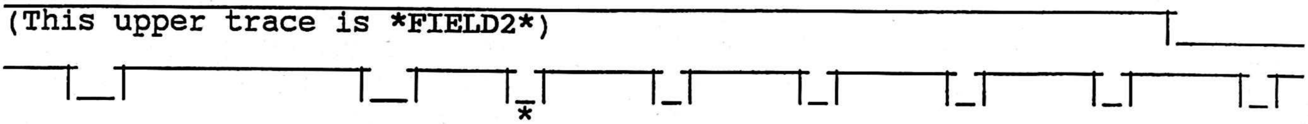
The 37-nanosecond one-shot is supposed to fire for 37 nanoseconds after a rising edge of *BLANK*. Trigger on the rising edge of U21-2, and look at U21-4. U21-4 should go high a nominal 37 nanoseconds after U21-2, but anything in the range of 23 microseconds to 50 microseconds should work. For safety and cleanliness, it would be nice if the time were somewhere between 32 and 42 microseconds. If it is not in this range, R17 will have to be adjusted.

The 145-microsecond one-shot is a little harder to see, since it is allowed to come to the end of its cycle only once each field. Externally trigger your oscilloscope with vertical drive from your sync generator. Look at *SYNC* and at U21-12. Ideally, U21-12 should come up exactly half-way between the fifth and sixth equalizing pulses (the first equalizing pulse is the first pulse that is only 2.3 microseconds wide. Normal sync pulses are 4.7 microseconds wide.)

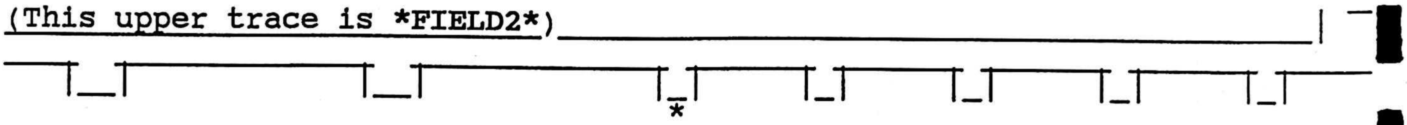
Nominally, that occurs 145 microseconds after **BLANK** goes high. In principle, any time from about 130 to 160 microseconds should work. Let's try to keep it in the range of 135 to 155 microseconds.

The last thing that we have to verify is that *FIELD1* and *FIELD2* are being determined properly. Set up your scope to externally trigger on vertical drive. Put one probe on *FIELD2* and the other on *RSYNC*. If you then set the horizontal sweep frequency to something on the order of 2 milliseconds per centimeter, you should see one vertical interval starting at the left side of the scope, and another vertical interval about three-quarters of the way across. If your scope has a HOLDOFF control, make sure that it is in the NORM position.

Video field one ends with one-half of a visible line. If your scope has locked up so that your vertical drive is at the start of field one, this is what you will see (in tiny detail) at about the 16.67-millisecond point on the *FIELD2* and *RSYNC* traces: (In both diagrams the "*" indicates the first equalizing pulse.)



If your scope happened to lock up on the beginning of field two, then you will see the following pattern and *FIELD2* should be low:



ADJUST THE FLOPPY DISK TIMING.

Before continuing any further, this is a good time to adjust the tweaks for the floppy disk. Turn to sheet 4 of the schematic and review the test procedure:

1) Make sure there is no "ENP" jumper.

2) Open the "TEST" jumper.

3) Hit the RESET button.

4) Close the "TEST" jumper.

5) Adjust "WPW" potentiometer [VR1] so that the "WPW" test point [TP2] is 300 nanoseconds.

6) Adjust the "RPW" potentiometer [VR2] so that the "RPW" test point [TP3] is 250 nanoseconds.

7) Adjust the "VCO" capacitor [VC1] so that the "VCO" test point [TP1] is 500 kilohertz. That is, of course, a 2-microsecond cycle time.

Make sure that you remove the "TEST" jumper, and you are done with the floppy disk for now.

REFRESH TIMING and MEMORY CYCLE TIMING.

This procedure will test bunch of things on sheet 2. First, look at U15-2 and verify that it never goes low. It shouldn't, because nobody is pulling U15-4 low (it should be floating, since the DISPLAY board is not connected.) If U15-2 is going low, then temporarily tack U15-4 to +5V. This should get rid of U15-2 going low.

Next, press the STEP switch - this should stop the 68000 from doing any accesses if perhaps you put some PROM's into their sockets.

Okay - we have locked out the CPU and made sure that no *REQTOP*s are coming through. The only thing that should be happening now are refresh cycles.

Look at *REQREF at U15-5. It should be pulsing low for about 0.4 microseconds every 10.24 microseconds. If it never goes low, check out the counter circuit at the top of the page. If it goes low and never comes high again there are a number of things you should check.

1) Make sure you have CLK at U15-9.

2) Look at *PRIEN* at U15-1. It should be low all of the time, except when it goes high for 320 nanoseconds while the refresh cycle is in progress.

3) If *PRIEN* is high all of the time, make sure that PAL3 at U54 and the eight-bit latch at U37 are interconnected properly. (You may have to buzz out the circuit board fixes.)

   Find out what state the PAL3 is in. If the states are changing, then MCA, MCB, MCC, and MCD should be following the pattern in the little table. See if DE0, DE3, DE5, and DE7 are doing anything. If not find out why. Don't that assume the trouble is in the PAL - it probably isn't, and they are too expensive to just throw away while trying to find a solder splash. Be sure that you understand the Boolean logic of the PAL3 (as found in the Appendix) and don't go throwing a PAL until you can demonstrate that it is generating the wrong output in response to certain set of inputs.

4) If the PAL3 seems to be stuck in a wrong state, or is cycling through a series of incorrect states, again, double-check the interconnections between U54 and U37.

I am sorry about being so vague here, but particularly with this circuit (and on the board version 4357-001-2 this is all post-fabrication alterations) there are just too many things that can go wrong for me to attempt (sitting at my desk, no less) to try to detail them all.

Anyway, let's assume that the refresh cycle is occurring every 10.24 microseconds, like it is supposed to. Let's double check the timing of the signals involved.

CHECK MEMORY CYCLE TIMING.

There is one extremely critical part in the memory cycle timing - that is the F379 at U15. We have discovered that this part is very finicky with respect to its enable-to-clock setup and hold times, so these need to be checked carefully. We have also discovered that very minor changes in waveform can affect it a lot. Check these times carefully.

First check the time from U15-1 rising to U15-9: Synchronize on the rising edge of U15-1. Look with your other probe (you do have two probes, don't you?) at U15-9. The time from U15-1 going high to U15-9 going high must be at least 6 nanoseconds - in this design it should always be greater than 10 nanoseconds, and will usually be even greater. It should not be more than 30 nanoseconds, and 34 is the absolute top limit.

Next, check the time from U15-1 falling to U15-9. Switch your triggering so that you are synchronized to the falling edge of U15-1. The time from U15-1 going low to U15-9 going high must also be at least 6 nanoseconds. If the AS574 at U34 is a worst-case part (9 nanoseconds) and the 16L8A at U54 is worst-case (25 nanoseconds), then together they use up 34 of the 40-nanosecond cycle, leaving only the minimum 6 nanoseconds. This probably won't occur, and is theoretically okay if it does occur. If we have trouble with this in the future, we may have to go for the more expensive 15-nanosecond 16L8B parts for PAL3. In any event, this time should be between 10 and 30 nanoseconds, just like the other one.

Even if the cycle is working, let's double-check the timing - it might save trouble later.

Within 40 nanoseconds of ***PRIEN*** [U15-1] going high, ***DE0*** [U37-15] should go low. Call the falling edge of ***DE0*** time zero and verify the following times:

DE0 is 40 nanoseconds wide.

The falling edge of ***DE3*** at 120 nanoseconds - rises 40 nanoseconds later.

The falling edge of ***DE5*** at 200 nanoseconds - rises 40 nanoseconds later.

The falling edge of ***DE7*** at 280 nanoseconds - stays down indefinately.

DE345 falls at 120 nanoseconds, comes up at 240 nanoseconds.

DE456 falls at 160 nanoseconds, comes up at 280 nanoseconds.

RAS falls at 40 nanoseconds, comes up at 240 nanoseconds.

CAS falls at 80 nanoseconds, comes up at 280 nanoseconds.

Once you have verified that all of these times are correct, then you can think about trying to get the 68000 running out of PROM, which is the next step.

TEST PROGRAM "OUT00":

Okay, you have the clocks running. Kill the power, and put the two 2716's which contain the program labelled "OUT00" into the PROM sockets. Check and double-check that you have placed the "0" and "1" PROMs into the proper sockets. Check and double-check that you have set up the jumpers for 2716 PROMs. (It is most distressing to think that there is something wrong with the CPU board, only to discover that you have a jumper in wrong.)

Turn on the power. If all is well, then you should be able to see a string of output pulses on U49-15, which is where ***OUT00*** pulses. These should occur every 4.5 to 5 microseconds (the time is not exact because of refresh cycles) and last exactly 120 nanoseconds.

If this is what you see, then you should proceed to the serial I/O test.

If you don't see these pulses, even after repeated applications of the RESET button, then we have to debug the 68000 and PROM circuitry.

The best thing to do at this point is to plug an address/data buss display box into the ribbon connectors P2 and P3. This is a simple little whatsit that just shows you, with LEDs, what the condition of the data buss and address buss is.

Once you have that box plugged in, then start the 68000 up in STEP. This requires a little bit of manual dexterity. First, you have to push the RESET button. While the 1.5-second RESET pulse is active you must push and hold down the STEP switch - you have to be holding it down when the RESET pulse finishes. This will cause the 68000 to hold in its very first memory cycle - the one that reads the sixteen bits of data at location 0000000. Let's look at the program that is in the PROMS:

```
0000000  01234    DW   01234      ;High-order stack pointer.
0000002  05678    DW   05678      ;Low -order stack pointer.
0000004  000C0    DW   000C0      ;High-order start address.
0000006  00008    DW   00008      ;Low -order start address.
```

Once those values have been accessed, the 68000 will execute out of PROM:

```
0C00008  033FC    MOVEW $00,0FFFF00   ;Hit location 0FFFF00 with 000.
0C0000A  00000
0C0000C  000FF
0C0000E  0FF00
0C00010  04EF9    JMP  0C00008        ;Jump back and do it again.
0C00012  000C0
0C00014  00008
```

When you actually step through this program, you will discover that the write access to location 0FFFF00 occurs immediately after the read of the JMP instruction at location 0C00010. This is a cute little "prefetch" that the 68000 does in an attempt to speed up processing.

That, of course, assumes that you are able to single step. Let's look at few possibilities.

When you hit RESET, and then hold the STEP switch, the data and address busses should all go high for about 1.5 seconds. Then, the address buss should go to 0000000 and the data buss to 01234.

If the address buss does not go to 0000000, then make sure that the ***RESET*** and ***HALT*** signals behave properly right at the pins of the 68000. Also make sure that **68CLK** can be found at pin 15 of the 68000, and that it is a nice, clean 12.5MHz.

If the address goes to 0000000, but the PROM data is wrong, then make sure that you have the right PROMs, that they are properly inserted into the proper sockets, and that the jumpers are set up for 2716s. If all of that is okay, then make sure that the ***PROMEN*** signal is low, which should make the **PROM** signal go low, which should enable the PROMs's AS573 data buss drivers. Make sure that the addresses at the PROMs match the address on the display.
If the address stays at 0000000 when you repeatedly press the STEP button, then the problem is either in the single-step hardware, (perhaps ***AS*** is not getting there, or some such thing) or the ***DTACK*** is not being generated. If you suspect the problem is with ***DTACK*** then you may become suspicious of the PAL1 that generates it. Go ahead and swap it with another if you like, but don't throw it away - they are just too expensive to be tossed unless you can prove it is no good. If the machine just comes up with address 000000 and sits there, you may find a logic analyzer helpful - look at things like ***AS***, ***REQCPU***, ***CPUCY***, ***DE5***, and whatnot - the question is, is ***DTACKOC*** being generated, and if not, why not. If it is being generated, then why isn't it getting to the CPU.

A distinct possibility is that the PROM accesses work, but that the program sticks on the OUT00. This again is a ***DTACK*** problem - examine the Boolean logic for the PAL1 and find out why you are hanging up.

An annoying problem that we used to see in the prototyping stage but have not seen much lately is the one where it single-steps fine, but goes into never-never land when you try to run it. (Never-never land is caused by a foolish data access during the jump, for instance. This could send the CPU off to non-existant memory space, eventually resulting, in most cases, in a HALT condition, which sends all of the data and address lines high.) This was usually caused by somebody besides the PROM drivers stomping on the data buss during PROM time - again, a logic analyzer can be useful if you suspect this: Trigger on PROM and other data buss driver enables.

Enough of that. Let's assume that you have the "OUT00" program working. The next step is to get the serial I/O cooking.

INITIAL SERIAL I/O TEST.

Kill the power and insert the "ECHO/DOTS" proms "0" and "1" into their sockets. This program does a simple exercise on RS-232 channel A. About once each second it sends an "A" out on channel A. The program also listens to channel A. Whenever it gets a character it immediately sends it back out again. This quickly determines that channel A is working well enough for you to proceed. Note: if only some characters come through on echo, make sure that you terminal is set up for the proper parity and stuff.

The endless question: What if it does not work? The first likely possibility is that we are hanging up waiting for a *DTACK* from one of the I/O codes to the DEUCE - these codes are handled in a different way than OUT00 - the *DTACK* is generated later. If you are hanging up on a DEUCE access, then look at the DTACK circuitry and the Boolean logic for PAL1. Examine the se-reset flip-flops and the F163 counter and see if that's where the problem is.

If the program is running, but you see no serial output, then start looking around the DEUCE. Are you getting *RDPULSE* and *WRPULSE*? Are the CS3, CS2, CS1, and C/D lines of the DEUCE being driven properly? Is the bi-directional driver being controlled correctly? Is the 1.8432MHz oscillator oscillating at 1.8432MHz? Is the serial data going from the transmit pin of the DEUCE? Is it making it through the 1488? Is the terminal hooked up properly?

If the terminal is displaying the one-second "A"s, but does not echo, make sure that you see the keyboard data at the input pin of the 1489, and at the receive pin of the DEUCE. Check and make sure that the LS175 got reset to all zeroes, so that your RS-232 terminal is being looked at on the receive end.

TEST THE DOT-MATRIX MEMORY.

Now that you have the terminal working, you can go to the next step and get the dot matrix memory working. First, let's just look at some of the timing signals involved.

First, check for **USDOT**. Externally trigger your scope with horizontal drive. **USDOT** should be a 25MHz clock that goes away for a while during each video sync interval. When it comes back, it should be locked up fairly closely to video - when you look at it while triggering on horizontal drive, you should see each **DOTCLK** cycle occurring in the same place, with a 10-nanosecond jitter. If you see it start up wrongly, you will have to trace out the ***KILDOT*** circuitry and find out what the problem is.

Next look for ***HOLDOFF***. It should go low during horizontal blanking, and end about one-half of a microsecond before the start of active video.

Another quick check of some basic timing, and we can actually look at some dots. Look at U86-3, which is the ***DDE0*** signal. If you synchronize on horizontal drive, you should see a steady stream of pulses on this line, except for a short time during video sync. These pulses should occur in the same place each line, with a 10-nanosecond jitter. If there are pulses occurring in the wrong places, you will have to check out the start-up logic of the F161 at U79 and the F379 at U85.

Leave one probe on ***DDE0*** and look at ***DDE7*** with the other. Make sure that the last ***DDE0*** (the one that occurs just before video blanking starts) is followed by a ***DDE7***. We have tracked down some weird problems that were caused by dot-matrix memory cycles being interrupted in the middle as blanking occurred - save everybody trouble by making sure that you have one, and only one, ***DDE7*** 280 nanoseconds after each ***DDE0***.

Another thing that should be checked at this point: Look at one of the memory chips - clip onto and synchronize to any RAM pin 4. This is the ***DDRAS*** signal. It should be there, every 640 nanoseconds. Clip also onto pin 5, which is the lowest order address bit.

It is very important that the RAS address hold time, which is 20 nanoseconds, and the CAS address setup times, which is 0 nanoseconds, be met. This delay is adjusted by the ALS08s and the ALS138. What would be ideal here is if the time from *DDRAS* going low to the time the address bits change is in the range of 25 to 30 nanoseconds. Very strange memory problems can result if this timing is off.

Okay - your basic timing signals are here. Let's look for some video.

First of all, hook up a black-and-white monitor up to the dot-matrix video output. We have found that when you power-up this board, it usually comes up in a pattern of horizontal stripes. Hopefully, that is what you are seeing now.

If you aren't, start tracing things back. Check the *RSYNC* and *RBLANK* inputs to the 75451s. If they are okay, look at where the dots come into the 75451. Are they there? If so, you will have to look forward to the video. If not, keep working back.

Is the flip-flop always low? It should be set at the beginning of each field - in NTSC, it should never go low. Are dots coming into U97-12? If not, check the two F323s.

Are they getting an **USDOT**? If they are, are they also getting hit with the **DDLOAD** pulses? (These should occur throughout active video.) At this point, if you still don't have any video, you have to go to the RAMs.

The main thing with the RAMs is to make sure that they are getting ***DDRAS*** and ***DDCAS*** signals. Since the 68000 is not accessing these memories, there should be two ***DDCAS*** pulses for every ***DDRAS*** pulse. The **DDLOAD** pulse should occur towards the end of the ***DDCAS*** pulse that occurs with a ***DDRAS***.

If you still are not seeing video, you had better make sure that the two AS867s that generate the display address are being reset and clocked properly. If they are, you will have to trace back the circuitry that does the address multiplexing and make sure its okay. If that does not show you anything, then maybe, just maybe, memory has managed to clear itself to all zeroes. On the other hand, probably not. Better you go back to the video connector and start working back again.

At this point, I will assume that you have some stable pattern of dots on your video screen. If you do, then it is time to see if the 68000 can read and write the dot-matrix memory.

Make sure that the ECHO/DOTS PROMs are in the PROM sockets. Hit RESET, make sure that you are echoing properly, and press "!" - this will transfer you from the "ECHO" program into the "WALKING DOTS" program.

This program runs through all of dot-matrix RAM filling it with 08000s. It then reads back all of those locations. If it gets an error, it tells you by writing an error message at the bottom of the terminal display. It then repeats, using 04000 as data, then 02000, etc. What you should see is a pattern of vertical lines shifting to the right. The program will tell you at the top of the terminal screen how many cycles have gone by.

If the program runs with no errors but you don't see the picture on the screen changing, then you may have some problem with the 68000 address multiplexing out to the dot-matrix RAMs. (The 68000 thinks that it is writing and reading all of dot-matrix memory, but it really is only hitting one location.)

If it runs with errors, you will have to go through the classic memory debugging problems: Is it shorted data bits, is it open data bits, shorted address, open address, is RAS or CAS missing on a chip, is it a bad chip, etc., etc. Double-check the RAS address hold and CAS address setup times - if these are wrong, then very odd things can happen.

Another source of very odd memory problems are glitches on RAS or CAS. If either RAS or CAS ever has narrow low-going pulses or spikes on them, it can cause the memories to forget. Synchronize on ***DDRAS***, turn off all the lights, turn up the oscilloscope intensity and really give it a good squint. Then do the same for ***DDCAS***. The most likely cause of such screwups is a foul-up in the video-blanking-caused disabling of the dot-matrix ring counter (U79 and U85). The last cycle is supposed to finish out, ending with DDE7, but we have had trouble with this during development.

Another likely source of trouble is lack of ***DTACK***. You try to get at the dot-matrix RAM and it just shuts down - actually, it is likely to go into never-never land. If you don't get a ***DTACK*** then the Buss Error circuitry will fire. When this happens the CPU will go looking into some memory location near 0000000 for an exception vector. Since we have no memory down there yet, this is quite futile.

If you absolutely never get a ***DTACK*** you should just trace out the circuitry starting where ***DISPADD*** and ***CPUCY*** feed into U92. Work through it to where you should get a **DTACKOC***. A logic analyzer may be of some use.

A logic analyzer will definately be useful if the test runs for a while and then stops dead. Hook up the logic analyzer to trigger on Buss Error. (You may want to tack a 4K resistor across the 20K timing resistor - this will make the buss error delay shorter and let you use a faster sampling time on your logic analyzer.) Use the other logic analyzer signals to probe out what is going on in the three flip-flops.

If you are hanging up waiting for a ***DTACK*** but not when you are trying to do a ***DISPADD*** access, then the most likely problem is that the AS574s at U106 and U105 are trying to drive the data buss when ***PROM*** is supposed to be driving the data buss. Again, a logic analyzer might be valuable.

A trick that we found useful when having ***DTACK***-related problems: Get rid of buss errors by lifting U51-9 or U51-10. This prevents Buss Error processing - if you hang up waiting for a ***DTACK***, the offending address is left on the address buss where you can see it.

Well, at this point you should have a working "WALKING DOT" test running. Let it run for a few minutes, long enough so that you are pretty sure that it really is working. Once that's done, you are ready for the next step, which is to plug in the "DOT-MATRIX

OPTIX" PROMs.

INSTALL DOT-MATRIX OPTIX.

Kill the power. Plug in the "DOT-MATRIX OPTIX" PROMs "0", "1", "2" and "3". Set up the PROM jumpers for 27128s. Turn on the power and you should be rewarded with the initial OPTIX display. You will also see the top part of the dot-matrix display fill up with the characteristic OPTIX RAM pattern.

Verify that all is well with this optix by running an X-test in the dot-matrix memory: x0C90000,0C9FFFF. Run it for at least 100 cycles.

We have never had any trouble making OPTIX work once the PROMs, the DEUCE, and the dot-matrix were independently tested. If OPTIX does not come up, then you probably have the PROMs in backwards, or the jumpers set wrong. The one new thing going on here is that both banks of PROM are being used, not just one. Use the visual indication of what is happening in dot-matrix RAM as a clue. Running and single-stepping with the display box hooked up may also give you some useful information.

Congratulations! Once you have checked the timing signals and gotten OPTIX running, you pretty much have tested everything that can be tested on the CPU board by itself. It is now time to take the board over to a test jig arrangement and check out everything else.

CPU FINAL TEST.

We will use a test setup to finish up the testing of the CPU board. Why? The answer is because we are trying to make the testing as sensible as possible. We are going to hook the CPU board up to a good sync generator, a good encoder, a good floppy disk drive, a good terminal (with a special 5-way test cable), a good Iomega drive, and a known good DISPLAY board. If you take all that, plug in a new CPU board, and it does not work correctly, then you can be fairly sure that the problem is in that CPU board. You can be particularly sure if that test setup works with a known good CPU board.

Remember - we have just tested out almost everything on our new board. We wouldn't try it out in the test setup unless we were fairly sure that it is going to work - that test setup is going to be busy. It will also be used for checking out DISPLAY boards, and will probably be used for the incoming inspection of Iomega disk drives and the like.

Anyway, let's test the board. Put the "SIN 1234" Software Identification Number test PAL into its socket and put your new CPU board into the jig. Power it up. You should be in OPTIX. If not, check your power supplies. Make sure the three interboard connectors are in properly. Does the CPU board go into OPTIX if you disconnect any or all of those three connectors? And so on.

Once you are in OPTIX do an X-test of locations 0020000-0040000 for at least one hundred cycles or so. Once that works, set up an X-test of locations 0000004-03fffff. Watch it for a few minutes, so that you are sure it is working, and then go get a cup of coffee or something. Let it run for ten cycles or so. Get out of it by hitting RESET.

At this point, you should not be getting any memory errors at all. If you are, double-check the ***RAS*** and ***CAS*** timing at the memory chips - trace it back if wrong.. If they are okay, try to figure out if the problem is a write error or a read error. (Write errors are persistant. Once the data goes in wrong, it stays wrong. Read errors tend to come and go. Refresh problems and RAS/CAS timing problems are characterized by a steady deterioration of the memory.) Then check out the signals which drive the DISPLAY board's address drivers (***GRTCPU***) and the data buss transceivers (***RAMEN***, ***WR*** and ***DSRD***).

One thing more to check with memory - does the Buss Error detection circuitry work? Check it out by looking at some memory that does not exist:

```
m03ffff0<return>
```

This should generate one line of memory data, followed by a "0400000:", at which point the cursor appears on the bottom line.

Okay, memory works. That means we can try to run the diagnostic programs that are kept on Iomega disk. Put an Iomega 20K program disk into the left-side drive, and type "lg<return>".

We now are testing out the Iomega interface for the first time. I can't say much about the failure modes - we have never seen it fail, except when somebody forgot to plug in the 50-pin ribbon cable. I suspect that it will either work, or not work. Your best bet in debugging is to pay attention to the error messages. Failing that, you probably should consider hooking up a logic analyzer to the Iomega control signals - do the "lg" and see what happens. Again, the Iomega manual will prove most helpful. Don't overlook the possibility of a shorted or open data bit - the drive will try to bitch about parity errors, but a screwed up bit may foul up the error response.

If the Iomega works, you will be confronted with the 20K program. The first thing it will do is display for you the contents of the Software Identification Number PAL - it should be 1234. It will then attempt to verify that interrupts work - it will announce that it is testing the field interrupts, and it will check that the activity of its field bit input is working properly. It will also check a range of values of N in the N-line interrupt circuitry. It will then try to load some fonts and animations.

Meanwhile, a set of color bars should have appeared on the color monitor. Once you have "hit a key when ready" you should enter the Diagnostic Mode.

In the diagnostic mode you will be confronted with a menu. The first thing to do is the Iomega test, just for completeness. This test will find an unused area of the disk and make sure that it can read and write there. The error messages will tell you if anything is wrong - a very unlikely occurrence, since the Iomega is already working.

The next thing you should do is a floppy disk test. Get hold of a blank floppy disk and use the diagnostic routine to format it. After that, run the floppy disk test. Like the Iomega disk test, it finds an unused area, writes on it, and checks it back. If you can format, read, and write a floppy then the floppy circuitry is probably in good shape.

The floppy circuitry is a bit more complicated, and probably a bit more error prone. If you have trouble, double-check the "WPW", "RPW", and "VCO" times. Make sure the "TEST" jumper is open. Make sure that the "ENP" jumper is open. There are a couple of diodes in the floppy circuitry - make sure they are in properly. Take a look at the one-shot - make sure that it is firing. On the 2793, make sure that the ***RDPULSE*** and ***WRPULSE*** signals are arriving and are sensible in timing. If in the future we find that the floppy is causing lots of trouble, we will expand the diagnostic procedures.

The last thing on the CPU board that needs to be fully tested is the serial I/O circuitry. To do this you need a special 5-plug cable. RS-232 channel A goes to the ADM-12 terminal. Channel B sends to C, C sends to D, and D sends back to B.

The RS-422 channels are looped around four-wise: A to B, B to C, C to D, and D back to A. The program will first check the three possibilities of the RS-232 channels, and then check the four possibilities of RS-422. It will also determine for all of these possibilities that the ***DEUCERDY*** signal is behaving properly. If anything goes wrong, the program will let you know.

If something does go wrong, you can activate the 4-way echo mode. This program behaves very much like the PROM program we described earlier. To fully check out any RS-422 lines that are causing trouble you will have to use the RS-232 to RS-422 converter that we have built into the display boxes. Don't forget that you get out of the 4-way echo mode by hitting Ctrl-C on any of the four inputs.

In other words, first run the 5-way cable test program. Suppose that you get the "RS-422 Channel B to Channel C no good" message. You then invoke the "RS-422 Echo routine. Use the RS-232 to RS-422 converter in the display box. You can hook up to channel B and see if it is transmitting okay, and hook it up to channel C and see if it is receiving okay. Once you have that figured out, it will be a simple matter to figure out what, exactly, is causing the problem.

One last test, and we are done. Invoke the four tests "Gray Scale", "Red Scale", "Green Scale", and "Blue Scale". See that they behave properly on the RGB monitor. Don't bother checking the voltage levels: that's a DISPLAY board test, and we are looking at the CPU board. But this does definately determine whether or not the palette accessing and the output strobes to the DISPLAY board are getting there properly.

At this point you have a working board. If nobody is going to be using the test setup for a while, let an animation run for a couple of hours.

That's it. You now have one completely checked-out CPU board. Take it out of the test rack (don't forget to pull out the test SIN PAL) and put it on the shelf. Later on, when you have built and tested the display board, you can take this (or anybody else's) CPU board out of the rack, marry it with the DISPLAY board, and with a minimum of testing and adjusting have yourself a working 10K/20K.

Welcome to the checkout sheet, which is a copy of the manufacturing documentation for the the 10K/20K CPU board.

At Dubner Computer Systems, somewhere in a file cabinet, we will have one of these documents for each and every 10K/20K board we ever build. It is critical that the revision level indications be kept up to date. This will make it possible for us to figure out what the hell is going on when somebody outside, (whom we may never have even heard of before, since these things are going to be sold through distributors), calls us up and says, "My 10K keeps drawing little obscene pictures, and I don't know how to make it stop." If we can ask him what the serial number of the boards are, and look up just what changes were made to them and when, we will be in much better shape than if we have to ask, "How many wires are tacked onto the back of the card, and what color are they?"

So the documentary information at the top of the first sheet is extremely important - as boards are upgraded by making later revisions, we really must make an effort to indicate it here. If we send a set of ECOs out to a customer to do himself, we had better make a note to that effect also - it could save a lot of trouble.

The rest of the data sheet is mostly supposed to help out people who are new to debugging 10Ks - it really does look a little like a Heathkit assembly manual. An experienced technician will know what steps can be safely skipped. An individual new to these boards, however, will probably find it very helpful to go through them step by step. If an effort is made to understand why each signal is being tested, the neophyte will find that he or she is learning a lot about these boards very quickly.

We expect that field service people will also find the checkout list useful. Granted, we have some test programs and test equipment here that they might not be able to get their hands on easily, but it is hard to think of a problem that would ruin a working board but at the same time manage to slip by without affecting any of the signals that we want checked.

CPU BOARD 4357-001-2 Revision Level ______ Serial number ____________

Assembled by: ____________________ Date: ____________

Stuffed by: ____________________ Date: ____________

Checked out by: ____________________ Date: ____________

Upgraded to level ________ by ____________________ Date: ________

Upgraded to level ________ by ____________________ Date: ________

Upgraded to level ________ by ____________________ Date: ________

Single-board test:

+5V Supply Voltage (4.9V to 5.1V): . . . . ________

-5V (-4.75V to -5.25) . . . . ________

+12V (11.4V to 12.6V) . . . . ________

-12V (-11.4 to -12.6) . . . . ________

Check CLK - Should be stable 40-nanosecond clock . . . . . ( )

Check 68CLK - Should be stable 80 nanosecond clock . . . . ( )

Check for *RSYNC* . . . . ( )

Check for *RBLANK* . . . . ( )

Check for *SYNC* . . . . ( )

Check for *BLANK* . . . . ( )

Check for *LINE* (40-nanosecond pulse as *BLANK* starts . ( )

Period of the 37-nanosecond one-shot? (32 to 42 nanoseconds) ________

Period of 145-nanosecond one-shot? (135 to 155 nanoseconds) ________

Is *FIELD2* correct? . . . . ( )

Make sure there is no floppy disk "ENP" jumper . . . . . . ( )

Adjust floppy test point "WPW" to 300 nanoseconds . . . . ( )

Adjust floppy test point "RPW" to 250 nanoseconds . . . . ( )

Adjust floppy test point "VCO" to 500 kilohertz . . . . . ( )

Remove floppy "TEST" jumper after adjustments . . . . . . ( )

Make sure that U15-2 is always high . . . . ( )

Hit STEP - what is *REQREF* period? (10.24 microseconds) . __________

What is the U15-1 rising to U15-9 setup time? (10 to 30ns) . __________

What is the U15-1 falling to U15-9 hold time? (10 to 30ns) . __________

(The following times are referenced to the beginning of *DE0*.)

DE0 should run from 0 to 40 nanoseconds . . . . ( )

DE3 should run from 120 to 160 nanoseconds . . . . ( )

DE5 should run from 200 to 240 nanoseconds . . . . ( )

DE7 should start at 280 nanoseconds . . . . ( )

DE345 should run from 120 to 240 nanoseconds . . . . . . ( )

DE456 should run from 160 to 260 nanoseconds . . . . . . ( )

RAS should run from 40 nanoseconds to 240 nanoseconds . ( )

CAS should run from 80 nanoseconds to 280 nanoseconds . ( )

Test program "OUT00" should produce a 120-nanosecond pulse on U49-15 every 4.5 to 5 microseconds. . . . . ( )

Test program "ECHO/DOTS" - is channel A okay? . . . . . . ( )

Does *USDOT* start up cleanly relative to horiz. drive? . ( )

How long is it from the end of *HOLDOFF* to the end of *RBLANK*? (About 0.4 microsecs) . . . . __________

Does *DDE0* start up cleanly relative to horizontal drive? ( )

Is there a *DDE7* pulse after the last *DDE0* pulse? . . . ( )

What is the *DDRAS* to row address hold time, right at the memory chips? (Should be 25 to 35 nanoseconds) . . . __________

Run at least 100 cycles of WALKING DOTS . . . . ( )

Install DOT-MATRIX OPTIX.

Run at least 100 cycles of x0C90000-0C9FFFF . . . . ( )

Make sure any changes you made for testing are removed (Buss Error, U14-4, etc.) . . . . ( )

These next tests should be run with the new board on a test setup with a known good power supply, a good Iomega, a good floppy disk drive, a good sync generator and encoder, and a good DISPLAY board.

Put a known Software Identification Number PAL into U6 of the CPU board.

Run 100 cycles of x020000,040000 . . . . ( )

Run 10 cycles of x04,03FFFFF . . . . ( )

Is Buss Error okay? (m03ffff0) . . . . ( )

Load in the 10K/20K program off the Iomega.

Does the proper Software ID number come up? . . . . ( )

Does it pass the interrupt and framing test? . . . . ( )

Get into the Diagnostic procedures mode.

Run the "Iomega Read/Write Test" . . . . ( )

Format a floppy disk . . . . ( )

Run the "Floppy Disk Read/Write Test" . . . . ( )

Run the "Five-Way Cable Test" . . . . ( )

Can you get red, green, blue, and gray scales? . . . . . . ( )

If the test jig is available, run an animation for a few hours . . . . ____________

Comments: (Use this space to indicate any special debugging problems you might have had with this board.)

OVERVIEW.

This section describes the suggested procedure for checking out a brand-new DISPLAY board. Unlike the CPU board, the DISPLAY board cannot be checked out on its own. However, it is a really good idea to use a CPU board that you know is okay when testing a DISPLAY board - otherwise you can end up chasing your tail.

Once again, this section is written mostly for technicians at Dubner Computer who are new to this circuit board, and as such may not be completely useful for field service purposes. But it may help.

This description uses the circuit designators of drawing 4357-002-2 Rev. C.

SETTING UP THE TEST SITE.

You presumably have a fully stuffed DISPLAY board and are ready to power it up. The same test jig you used for debugging the CPU board could also be used for working on the DISPLAY board. On the other hand, it makes a certain amount of sense for you to use a new debugged CPU in a new 10K/20K frame to work on your DISPLAY card. After all, if you can get your new DISPLAY card working with a new CPU board in a new frame, then you can run final tests on the thing and have a deliverable system. There is something to be said for that.

You will be needing an accurate voltmeter, and a 100MHz oscilloscope.

CHECK THE POWER SUPPLY VOLTAGES.

Whether you are using the laboratory test setup, or working with a newly assembled system at your bench, wire the whole thing together and turn it on. The first thing to do is to make sure the power supply voltages are okay: Check +5V, -5V, +12V and -12V.

CHECKING OUT THE BASIC MEMORY FUNCTIONS.

You will recall from the circuit description that the DISPLAY board has two major functions: The first is to behave like a 4 megabyte memory for the 68000, the second is to take specific information in that memory and convert it to video. We will first make sure that the memory is behaving properly.

I have been assuming that you have a copy of the DOT-MATRIX OPTIX PROMs installed. If all is working properly, then the machine should have come up in OPTIX.

If it does not, then unplug the three 50-pin ribbon connectors P1, P2, and P3. If OPTIX still does not come up then you have some kind of problem on the CPU board. If OPTIX does come up, then reconnect each of the three connectors one at a time, so that you can see which of the three is bringing the machine down. Trace out the signals accordingly.

If P3 is the culprit, check out the ***REQTOP*** line: if it goes low, and does not come up during ***GRTTOP*** time, it will lock out all 68000 accesses, since it is the highest priority. If ***REQTOP*** is okay (it will be bouncing up and down some) then you are probably dealing with some kind of shorted clock line or the like. If connector P2 is causing a problem you might have a shorted data buss line. It might also be some problem with the control signals going U6 and U7, which are the bi-directional data buss drivers. A problem involving P1 almost has to be a shorted address buss line.

CHECK RAS/CAS TIMING.

If OPTIX comes up running, check the RAS/CAS timing before you do anything else. Pick any of the 128 memory chips, and synchronize on the falling edge of pin 4, which is the ***CAS*** signal. Squint at it real close in subdued light - it should always be low for 200 nanoseconds - any short pulses or spikes are a real no-no. Do the same for the ***CAS*** pin.

Compare ***CAS*** to ***RAS*** - ***CAS*** should occur 40 nanoseconds after ***RAS***. Now check the RAS-address hold time, which is also critical: Synchronize on the falling edge of ***RAS*** and find an address bit which is changing with the second probe. The time from the falling edge of ***RAS*** to the time of the address change can be no less than 20 and no more than 40 nanoseconds for most dynamic RAMs. Make very sure that the timing is valid. Because each row of sixteen chips has its own ***RAS*** driver, and because there are four separate RAM address lines, it is a good idea to check this timing on all eight banks of RAM - just clip onto one of the sixteen chips in each row. It should only take a minute.

Once you have done this you are ready to run some memory tests.

Run a quick one first - x0110000,0120000. This should let you know quickly if things are okay. If they are not okay, there are a number of possibilities.

First, make sure that you are getting **68LTCH** at the AS573s.

As you continue, keep in mind that memory is organized so that 128 bits are accessed at a time. If a memory chip is missing (or a data line is no good, or the chip is not getting RAS or CAS), one bit will come up wrong and the last hex digit of the address will always be the same.

If a 16-bit bank of memory is missing *RAS* or *CAS* or *WR*, you will see the same pattern, but all 16 bits will be wrong. You will see this same pattern if there is something wrong with the AS573 data receivers, or with the signals that control them.

Decide if it is a write error, a read error, or a screwy error. A write error is characterized by the locations involved having the same data in them no matter how many times you read them. The inaccurate data in read errors tends to come and go. A screwy error is where the data is wrong, and when you look at it it gets wronger, and it steadily deteriorates after that.

One possible cause of screwiness is lack of proper refreshing. The other potential causes seem to involve glitches on RAS or CAS, or invalid address setup and hold times.

Let's assume that you have run the x0110000,0120000 for a hundred or so cycles with no problem. Do all of memory for a while.

If you are running DOT-MATRIX OPTIX, then do x4,03FFFFF for 10 or 20 cycles. If you are running with the 10K/20K Diagnostic Mode, then do x0100000,03FFFFF.

TEST THE DISPLAY.

At this point you should have absolutely no problem loading in the 20K program off of an Iomega cartridge. When the program starts up it is supposed to give you color bars - load up the program, and hold at the point where it asks you to strike a key. If you get color bars, then you may as well continue into the diagnostic mode. If you don't, then there are a whole lot of things that might be causing the trouble. I can't possibly attempt to detail them all - the actual diagnostic path to be followed can only come from experience in debugging these boards. I can, however, suggest that running through the basic timing signals is almost always fruitful.

Make sure that ***HOLDOFF*** is behaving correctly.

Make sure the ***DE345*** and ***DE456*** signals exist.

Make sure that ***FIELD*** causes its string of followers at U17. Make sure that there are no glitches during the field.

Make sure that you get ***LINEC1***, ***LINEC2***, ***LINEC3***, ***LINEC4***, and ***LINEC6*** every time ***SYNC*** goes low.

Verify that those pulses are making their way to the PAL1. Observe that pin 17 of PAL1 seems to do a low-to-high transition for every ***LINEC1***, as a single example. (If there are other problems that seem to be PAL1 related, you will have to check all the other inputs as well.)

Make sure that ***TLDHI*** goes low with ***LINEC4***, and that within a microsecond or two ***TLDLO***, ***TLD1ST***, ***TLD2ND*** and finally ***TLINE*** go low in succession. Observe that once ***TLINE*** goes low it stays low for the remainder of the line.

Look at the signals at the PAL2. Observe that pin 17 does a low-to-high during ***TLDHI*** and that pin 16 does a low-to-high during ***TLDLO***. Verify later on that both of those pins undergo low-to-high transitions at the end of ***DE345*** when pin 3 is low.

Make sure that the S195s that generate the ***TEx*** signals get parallel loaded during ***TLD1ST***. Verify that they get clocked first at the end of ***TLD2ND***, and subsequently with every ***TEOS***.

Make sure that the signals going into and out of U145 all seem to make sense.

Make sure that the 29521 setup time of 10 nanoseconds, and the hold time of 3 nanoseconds are both met.

Make sure that **TLOAD** and **TLATCH** both behave sensibly. (Look at the circuit description and the PAL4 Boolean equations for detailed information.)

Trace the various signals through the run-length counters. Make sure that ***TEOS*** is being generated at the right time.

At this point, since you have let the 10K/20K program put up color bars, then you should be seeing about sixteen **TCLOCK** pulses during each video line. This means that the **TCOLOR** lines should be should be changing. Trace the **TCOLOR** lines to the palette RAMs. If the RAMs look like they are being read-accessed correctly then the problem may be with writing them. If necessary, use OPTIX to write little test programs that hit the palette RAMs.

If the data is coming out of the RAMs, then trace it through to the D-to-A parts. Make sure the **VCLOCK** is okay. Make sure that voltage levels of the proper magnitude are coming out of the D-to-A parts. Trace the signal through the filter.

More help than this I cannot give you at this point.

RUN THE DIAGNOSTIC MODE DISPLAYS.

Assuming that you have started the 10K/20K program and that you have gotten color bars, enter the Diagnostic Mode.

Put up red, green, blue, key, and gray ramps. Make sure that the output voltage levels make sense for each.

Put up stripes of width 2. The screen should not break up at all.

Execute the Quick Warp function. If the pattern looks okay, then you have a properly functioning video generator.

Enter the 20K compose mode. If you can type characters with the cursor properly generated, then this checkout procedure is complete.

Let an animation run for a few hours.

On the following pages you will find a copy of the manufacturing documentation for the the 10K/20K DISPLAY board. A file copy of this documentation will be kept for every DISPLAY board we build.

Checkout procedure for DISPLAY board August 27, 1985

DISPLAY BOARD 4357-002-2 Revision Level ______ Serial number ______________

Assembled by:________________________________ Date:______________________

Stuffed by:__________________________________ Date:______________________

Checked out by:______________________________ Date:______________________

Upgraded to level ________ by ________________________ Date: ____________

Upgraded to level ________ by ________________________ Date: ____________

Upgraded to level ________ by ________________________ Date: ____________

+5V Supply Voltage (4.9V to 5.1V): . . . . ____________

-5V (-4.75V to -5.25) . . . . ____________

+12V (11.4V to 12.6V) . . . . ____________

-12V (-11.4 to -12.6) . . . . ____________

Does OPTIX come up running? . . . . ( )

Is *RAS* okay? . . . . ( )

Is *CAS* okay? . . . . ( )

What is the RAS to address hold time? (20ns. to 35ns.) . . ____________

How many cycles of x0110000,0120000 (at least 100) . . . . ____________

How many cycles of x4,03fffff (at least 10) . . . . ____________

Do you get color bars when you start up the 10K/20K . . . ( )

If you don't get colors bars, here are a bunch of signals that you might find are useful to check.

Is *HOLDOFF* okay? . . . . ( )

Do you get FIELD+0+1, FIELD+3, and FIELD+5 at U17? . . . . ( )

Do you get LINEC1 through LINEC6? . . . . ( )

Are TLDHI, TLDLO, TLD1ST, TLD2ND behaving properly? . . . ( )

Are the TEx signals loading and shifting at TLD1ST and 2ND? ( )

Are the 29521 drive signals at U145 okay? . . . . ( )

What is the 29521 instruction-to-clock setup time (10ns or more) ________

What is the 29521 instruction-to-clock hold time (3ns or more) ________

Is TLATCH behaving sensibly? . . . . ( )

Is TLOAD behaving sensibly? . . . . ( )

Are the TCOLOR lines changing at TCLOCK? . . . . ( )

Are the address lines to the palette RAMs following TCOLOR? ( )

Are the data lines from the palette RAMs changing? . . . . ( )

Is VCLOCK okay? . . . . ( )

Are the voltage levels coming out of the A-to-D chips? . . ( )

Are they making it through the filters? . . . . ( )

At this point, let's assume that you are getting color bars when you start up the 10K/20K program. Run the Diagnostic Mode functions:

Are the red ramps okay? . . . . ( )

blue? . . . . ( )

green? . . . . ( )

key? (Check this with a scope) . . . . ( )

gray? . . . . ( )

Put up stripes of width 2. . . . . ( )

Does the Quick Warp function display properly? . . . . ( )

Hour many hours did you run an animation . . . . ____________

Comments:

We will now attempt to learn how the SEK board works. The name SEK refers to the three sections of the board: the Sync Regenerator, the Encoder, and the linear Keyer. We will first examine the Sync regenerator.

THE SYNC REGENERATOR

First, what is a Sync Regenerator? A sync regenerator is a circuit which given a standard video signal (on this board we call that signal the genlock reference video), regenerates various video timing signals. This board regenerates SYNC, BLANKING, and SUBCARRIER. Let's start following the circuit through from the point where the genlock reference video comes onto the board.

GENLOCK REFERENCE INPUT AND D.C. RESTORATION

Looking at the top left of sheet 1, you will find the BNC connector which brings the genlock reference onto the board. You will note that the signal goes directly to a Grass Valley Group D.C. Restore chip. This device is wired up so that the back porch of the video signal is clamped to ground. The output of this part is on pin 14. For further information, you will need reference to Grass Valley's documentation on that part which, for the most part, is proprietary. For folks in the field, it either works or it doesn't. For Dubner people doing initial debugging, we have found two capacitors to be highly critical. If the C82200pF capacitor is no good for some reason, or if the 4.7uFC83 capacitor is no good or perhaps is in backwards, then the part will simply not work.

FOUR-TIMES SUBCARRIER PHASE-LOCKED LOOP

In order for this to work properly, the 14.31818MHz voltage-controlled-crystal-oscillator called VCX014 must be locked up to the subcarrier burst of the genlock reference video. This is done with a phase-locked loop. The 14.31318MHz output feeds through an F244 buffer into a two-stage shift ring counter formed by two elements of an F175. The two-stage ring counter does a divide-by-four which results in a subcarrier-frequency signal. One of the subcarrier-frequency outputs of the ring counter is fed back into an F86 exclusive-or gate.

The other input of the F86 is the output of a comparitor which is comparing the video signal to ground. Thus, during video burst, the output of the F86 is essentially a comparison of the incoming burst with the subcarrier being generated on the board. You will notice that the F86 uses a regulated 5-volt power supply--this gives the phase-locked loop circuitry immunity to fluctuations on the regular 5-volt supply.

The output of the F86 is averaged by an RC circuit and goes into a Harris 2425 sample-and-hold. The sample signal is called *BURSTGATE*. We will consider where *BURSTGATE* comes from later; meanwhile, just believe me when I say that it goes low during burst time. The 2425 is set up to amplify the voltage on the RC slightly. This signal then feeds into the control pin of the voltage-controlled crystal oscillator.

The end result of all this is that the incoming burst is compared to the on-board subcarrier. The comparison is latched and amplified by the Harris 2425 and is used to drive the oscillator which generates the on-board subcarrier. That loop is the phase-locked loop and it is constrained to lock up the incoming video. We end up with a four-times-subcarrier and a subcarrier signal which has a definite and repeatable phase relationship with the incoming burst.

BURSTGATE GENERATION

If you look back at the D.C. restore part, you will find that its output goes to two places. The first is the comparitor which generates the burst comparison. The second place is an operational amplifier which boosts the signal by about ten. The output of that op-amp goes to a second comparitor whose job it is to generate the *SYNCIN* pulse.

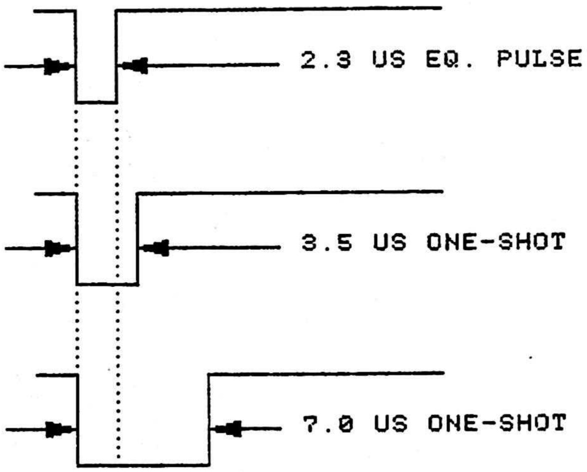
An examination of a standard video signal will show that the sync pulse has an upper level at zero volts and goes down about 0.3 volts. Thus, the output of our op-amp will go down to about minus 3 volts. The signal goes through an RC filter which is designed to get rid of some of the video subcarrier and then to the comparitor which is set to compare the video to about minus 2.5 volts--only sync should go down that far. The output of that comparitor is called *SYNCIN* and after being clocked through a 74F175 it goes to the Signetics 82S159 IFL.

The 82S159 is a fuse-programmable part which is similar to a PAL--the difference is that it is even more complicated. This particular IFL (IFL means Integrated Fuse Logic) is set up to perform two functions. The first function is to determine the type of sync pulse to regenerate based on the type of incoming sync signal.

If you look at a standard video signal, you will find that there are three different types of sync pulses: the standard 4.7uS horizontal sync pulse, the 2.3uS equalizing pulse, and the 27.1uS vertical sync pulse. In any case, the falling edge of any type of incoming sync pulse starts a series of events. The first thing to happen is the IFL brings the *PULSE1* signal low which then causes the 3.5uS and 7.0uS one-shots to begin firing. The IFL then waits until *SYNCIN* goes high again and then uses the time constants of the one-shots to determine the type of incoming sync signal. Let's examine each of the three types one at a time.

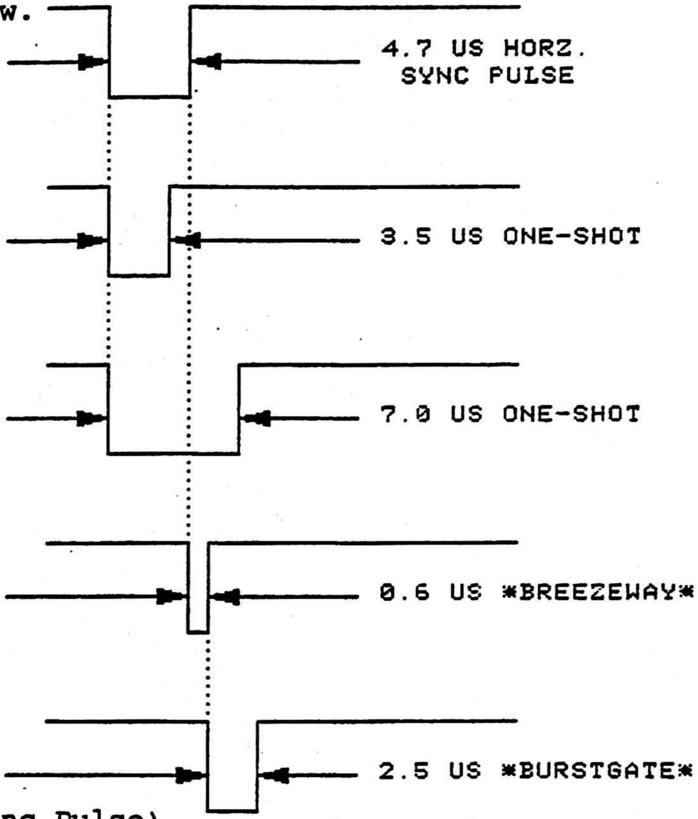
TYPE 1: 2.3uS Equalizing Pulse

If the incoming sync signal was a 2.3uS equalizing pulse, both the one-shots are still low when *SYNCIN* goes high again, as can be seen in the figure below



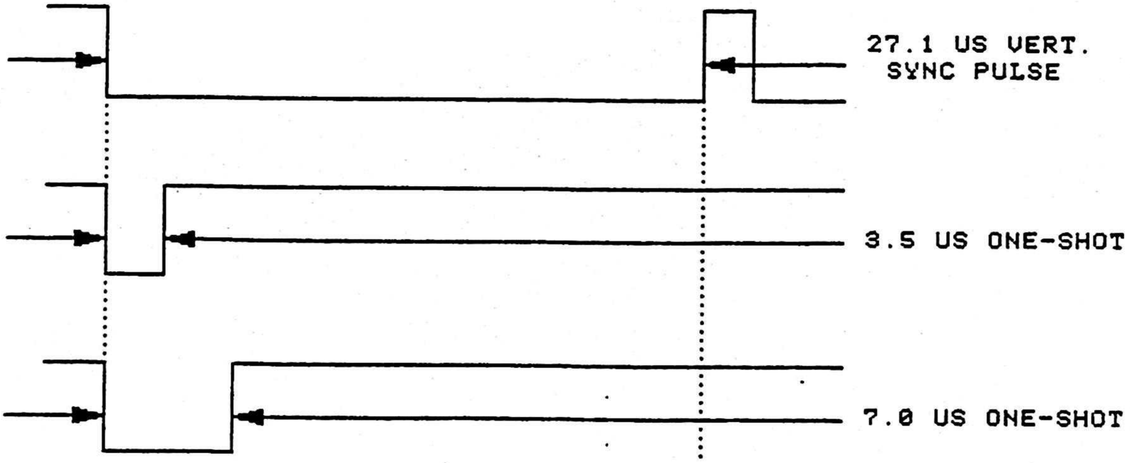
TYPE 2: (4.7uS Horizontal Sync Pulse)

If the incoming sync signal was a 4.7uS horizontal sync pulse, the 3.5uS one-shot is high and the 7.0uS one-shot is low when *SYNCIN* goes high again and horizontal sync pulses the IFL and brings the *BREEZEWAY* pulse low for 0.6uS. When the *BREEZEWAY* pulse goes high again, it fires off a 2.5uS one-shot which generates the *BURSTGATE* signal. The above sequence of events is illustrated in the figure below.


4.7 US HORZ.
SYNC PULSE
3.5 US ONE-SHOT
7.0 US ONE-SHOT
0.6 US *BREEZEWAY*
2.5 US *BURSTGATE*

TYPE 3: (27.1uS Vertical Sync Pulse)

If the incoming sync signal was a 27.1uS vertical sync pulse, both the one-shots are high when *SYNCIN* goes high again, as can be see in the figure below.


27.1 US VERT.
SYNC PULSE
3.5 US ONE-SHOT
7.0 US ONE-SHOT

DEFINING THE START OF A FRAME

The other job that the IFL has is defining the top of a video frame. It does this by looking for the first equalizing pulse by the means described above. When it finds an equalizing pulse, it knows that it is at the top of a video field. The question is: which field?

It answers that question by looking at the 40uS one-shot which gets fired off at the end of each *BREEZEWAY* pulse. If the first equalizing pulse occurs within 40uS of the last breezeway, then we are at the beginning of field 2. If the first equalizing pulse occurs more than 40uS after the last breezeway, we know that we are at the beginning of field 1, and the IFL yanks down the *FRAMESYNC* pulse for one CLOCK cycle (about 140 nanoseconds).

This *FRAMESYNC* pulse does two things. The first thing that it does is cause the three F163 counters in the bottom left of sheet 1 to be loaded with the values from the two binary switches. The value of these switches determines where the regenerated SYNC occurs relative to the incoming SYNC. The larger the number, the further to the left the regenerated SYNC is when compared to the incoming SYNC.

The job of these three counters to to keep track of where we are on each line. The outputs of these counters go into the 16L8 PAL which decodes the last cycles of subcarrier on the line with both inverted and non-inverted outputs. The inverted *LINEND* output goes into the synchonous clear input of the F163s. Thus, these counters get loaded with an arbitrary value at the beginning of FIELD 1, and then continually count up to the end-of-line where they are cleared to zero.

The positive-going *LINEND* signal goes into the other three F163s just above. The job of these counters is to keep track of which of the 525 video lines we are on. These counters get cleared to zero when *FRAMESYNC* goes low--they then count up by one every time *LINEND* goes high.

The outputs of these counters go into PROM1. This PROM takes in as input the line number that we are on and generates as output a 4-bit line type code. The line type contains a couple of pieces of inherent information. Line type 15 (all four bits on) is reserved for indicating "not-a-valid-line." Practice lines 526 through 1024 which are not used in NTSC video, are all line type 15. This causes the LS30 to clear out the line counter F163s to zero when line 526 is reached. The result of this makes it possible for the sync regenerator circuit to "free-run" properly--it generates legitimate SYNC and BLANKING signals even though there is no genlock reference signal.

The other piece of inherent information in the line type is the high bit, also called NOBURST. When this bit is on, it tells some circuitry further downstream that there should be no burst information following the SYNC pulse. More on this later.

The 4-bit line type gets synchronized up by being latched into an AS574, from which it gets sent to the top four address bits of PROM2 and PROM3. The bottom eight address bits of PROM2 and PROM3 come from the subcarrier cycle count of the three F163 subcarrier cycle counters.

PROM2 contains the SYNC information for each type of line, four bits at a time. PROM3, in similar fashion, contains the BLANK information. As an example, let's look at PROM2 (the SYNC information) for the very first line of a frame.

FRAMESYNC comes along resetting the line counters to zero. For simplicity, let's assume that the subcarrier cycle counters are also set to zero at the same time. The line count of zero gets sent to PROM1, which gives us back a line type of 8. Note that line type 8 means that NOBURST is high which makes sense: a line which is two equalizing pulses is not supposed to have burst on it.

The line type of 8 goes to PROM2, along with the subcarrier cycle count of zero from the F163 subcarrier cycle counters. PROM2 responds with 4 bits of zeroes which go to one-half of the LS153, which is a dual 4-to-1 multiplexor. The LS153 uses the bottom two bits of the subcarrier cycle count to sort out which of the four input bits is the current state of SYNC. That bit gets sent to the AS574 (so that it is synchronized with CLOCK) and from there to the transistor circuitry so that it gets rearranged as a video level. When the subcarrier cycle counter gets up to around subcarrier cycle 16 of this line, PROM2 will start generating ones, rather than zeroes, since sixteen bits of CLOCK frequency represents the width of a synchronizing pulse. PROM3 generates BLANK information in exactly the same way.

This concludes the section on the sync regenerator.

THE RGB TO NTSC ENCODER

Sheet 2 of the SEK drawing is devoted to the circuitry supporting the Motorola MC1377 RGB Encoder chip. The whole thing is relatively simple.

On the left side of the sheet you will find three identical circuits. These have the job of buffering in, and repeating out, the RGB signals from the DISPLAY board. Each signal is brought in, terminated to 75 ohms, multiplied by two and sent out again, presumably for use in an RGB monitor. The times-two gain is controlled by the first potentiometer in each circuit, and should be adjusted so that the output of each signal is 0.7-volts peak-to-peak into 75 ohms.

Once the first-stage gain is adjusted, the second potentiometer must be set so that the level of each signal going into the MC1377 is 1-volt peak-to-peak for full white levels.

The MC1377 also needs a SYNC signal. This signal comes from the adjustable delay line, and goes into pin 2 of the MC1377 with a range of 0-volts to about 2-volts. The RC does a little bit of softening of the edges so that the output approximates the 140 nanosecond rise-time requirement of video standard RS-170A.

In order to generate NTSC video, the MC1377 also requires a reference subcarrier signal. Back on sheet 1 we discussed the two-stage ring counter. This counter produces four different subcarrier-frequency square waves. A switch and an F153 decide which of those four signals is PHASE. PHASE gets sent to sheet 3 where it gets pumped into the transistor circuit at the top of the page. The 5K potentiometer is used to adjust the fine phase over a 90-degree range.

The final input needed by the MC1377 is a burst gate. In this design, the placement and width of burst is established by two one-shots. The first is nominally set for 5.3uS and places the start of burst with reference to the leading edge of sync as it goes into the MC1377. The second one-shot is nominally set for 2.5uS and determines the duration of the burst gate. The 75451 and the dual-diode circuit is designed to make the burst gate signal into the MC1377 vary between 0.4 and about 1.2 volts. The MC1377 is designed to insert burst only when the voltage on pin 1 is between 1.0 and 1.3 volts.

Of the gobbledegook to the right of the MC1377, I will say first that your best source of data is Motorola--most of that circuitry was copied directly from them. I will say this: luminance gets developed on pin 6, gets delayed by the delay line and then reinserted on pin 8. The chroma signal gets generated on pin 13, gets filtered by the transformer circuit and then reinserted into pin 10. The potentiometer between pin 10 and the transformer controls the chroma gain relative to the luminance. The potentiometers that connect to pins 11 and 12 are used to get rid of residual subcarrier in sync and black, and the potentiometer connected to pin 19 is used to fine-tune the R-Y and B-Y quadrature. The final video gets generated on pin 9, is divided by two with a couple of resistors, and departs to sheet 3.

LINEAR KEYER

Sheet 3 of the SEK drawing is mostly linear keyer, with a little bit of power supply thrown in for good measure. The +/- 12-volt to +/- 10-volt regulator is found on the right side of the drawing.

The left side of the drawing is dominated by the three Grass Valley Group D.C. Restore parts. The one on the top has the job of doing a D.C. restore on the video signal generated by the

MC1377--it is needed too. The MC1377's video comes out with a D.C. bias of about 4 volts. Once the signal is moved down to zero, it goes to the operational amplifier in the lower middle part of the drawing. Once it is boosted up to the level needed to drive 1-volt peak-to-peak into 75 ohms, it goes to two connectors. The BNC is for use by the world, the Coaxicon connector is used to bring the signal down to the display board where the cursor gets inserted for the PREVIEW display.

The 10KVIDEO signal is the foreground signal for the keyer. The background signal is D.C. restored by the second Grass Valley SC0423-00 and goes through an adjustable resistive divider. Both the background and 10KVIDEO signals, once they have been D.C. restored, go to the Grass Valley 2-input multiplier.

The two-input multiplier really needs three inputs--the third is a linear key signal. The key signal comes from the DISPLAY board and is D.C. restored by the third SC0423-00. Unlike the other four D.C. restore circuits, this one does not necessarily force the blanking level to ground. It rather restores to the level specified by the potentiometer on pin 2. This pot is used to set the low-level cutoff point of the multiplier.

Once the key signal is D.C. adjusted, it gets sent through a delay line which compensates for the delay in the RGB encoder circuitry. After that, it goes through the op-amp circuit which is used to adjust the key level to the 0-volt to 1-volt range expected by the multiplier.

It should be noted here that the key signal from the DISPLAY board always goes to zero during BLANKing time. The key circuitry is set up so that the signal from the background input is used when the key signal is zero. Thus, during blanking, the output of the key circuitry is entirely the background video, and nothing of the 10KVIDEO.

After making its way out of the multiplier, the mixed video goes into an op-amp circuit which conditions it for life as a 75-ohm signal. Final output gain is controlled by the last potentiometer--chroma balance is adjustable with the two adjustable capacitors on the multiplier and the op-amp.

And that is pretty much that!

In this section, we will go over the 10k SEK board element by element, describing what you should expect to see at each point.

We will start with sheet 1, which is the sync regeneration circuitry.

GENLOCK REFERENCE INPUT

Verify that all power supply voltages are correct.

Look at the output of the genlock reference D.C. restore part. It is easiest to see at the point where it goes into the burst-detecting comparitor. It should be a good clean video signal, with blanking level sitting just about on zero volts. Make sure that the output of the comparitor looks like a short burst of square wave during burst time.

Look at the input to the other comparitor. It should be about 10-volta peak-to-peak, with blanking level near zero. The subcarrier frequencies should be attenuated somewhat. The threshold of the sync pick-off comparitor should be well below the burst, and well above the bottom of the amplified sync level. Verify that ***SYNCIN*** makes sense.

PHASE-LOCKED LOOP

Verify that the on-board subcarrier and **CLOCK** are stable with reference to the incoming video. If not, you will have to trace out the phase-locked loop and see where it is broken. When we have had problems, they usually have been with too much gain around the Harris 2425.

IFL CIRCUITRY

Verify that the firing periods of the four one-shots are each within about 10% of the nominal periods. There is not much else that can go wrong with this part of the circuit. (I don't think that we have ever seen an IFL that was really bad.)

SYNC/BLANK GENERATION

The problems we have had in this part of the circuitry have always been due to open or shorted lines from the counters to the rest of the circuitry. Using an oscilloscope to make sure that they are all moving, and moving differently, has always been sufficient to get it working properly.

When following the circuit out to the RGB encoder, be aware that the 120 nanosecond variable delay line is fairly easily broken.

RGB ENCODER

Verify that the outputs of the R, G, and B, buffers are each 0.7-volts peak-to-peak into 75 ohms. (That is, of course, 1.4-volts peak-to-peak at the outputs of the op-amps.)

Verify that the inputs on pins 3, 4, and 5 of the MC1377 are each 1-volt peak-to-peak.

Verify that the SYNC input on pin 2 is 0-volts to 2-volts.

Verify that the burst gate input to pin 1 is about 0.4-volts to about 1.2-volts.

Verify that the subcarrier input on pin 17 is about 2 to 3 volts. Be aware that the 0-5K potentiometer varies the incoming subcarrier over about 90 degrees with more-or-less constant gain, then varies it for another 90 degrees or so with decreasing gain. In other words, play with it a little bit and make sure that you are on the constant-gain plateau, and not in the changing range.

LINEAR KEYER

Verify that each of the three D.C. restore chips is doing a D.C. restore.

Verify that the output of the key signal op-amp (at test point 1) runs from slightly below zero at blank level to just above 1-volt at the high level.

Verify that all three inputs appear on pins 1, 20 and 22 of the multiplier.

Verify that the output appears at pin 1 of the final mixed-video op-amp.

Verify that the final output is 0.7-volts peak-to-peak into 75 ohms.

Do the same for the 10K VIDEO output-buffering op-amp.

GPI PORT "D" 4+5

I. The CPU card is the bottom-most card. It has three (3) BNC connectors.

1) DOT MATRIX OUT. This output goes to any ordinary monochrome monitor. The monitor should terminate the signal with 75 ohms. All the status information will appear on this output.

2) BLANK IN. Video blanking for 10/20k system timing.

3) SYNC IN. Video sync for 10/20k system timing. When the machine contains a SEK card (three card system), BLANK IN and SYNC IN should be connected to BLANK OUT and SYNC OUT. The SYNC OUT of the SEK card cannot drive a 75 ohm load. If you wish to use an RGB monitor with external sync, house reference or some other sync source must be used.

II. The middle card is the display card. It generates red, green, blue and key signals. It also brings in a NTSC PGM IN signal and inserts a cursor signal over it. This signal is labeled CURSOR OUT and is connected to a color monitor. There are two (2) adjustments and six (6) BNC connectors on this card.

1) CURSOR POSITION. Adjusts the cursor position in reference to the PGM IN video to compensate for the encoder delay.

2) CURSOR GAIN. Adjusts the video level appearing at the CURSOR OUT output. This does not adjust the cursor level separately.

3) PGM IN. This signal comes from the PGM OUT of the SEK card or an external encoder. The cursor will be 'keyed' over this signal.

4) CURSOR OUT. This is the encoded 10/20k video plus the 'keyed' cursor signal.

5) KEY OUT. Goes to KEY IN on the SEK card or to an external keyer. This is not a simple on/off type of key signal--it has 256 possible levels. The SEK card has a linear keyer. This makes it possible to key anti-aliased characters over a NTSC video signal. If you decide to use an external encoder or an external keyer, then you must use an external delay line on the KEY OUT before going into your switcher.

6) RED OUT. Goes to RED IN on the SEK or to an external encoder.

7) GREEN OUT. Goes to GREEN IN on the SEK card or to an external encoder.

8) BLUE OUT. Goes to BLUE IN on the SEK card or to an external encoder.

NOTE: These outputs have 'set-up' digitally added to them. If you use an external encoder, this may produce a set-up level of about 15 IRE units. This may be corrected by following the procedure outlined in Appendix A.

III. The SEK card regenerates sync from a reference video source, encodes the red, green and blue signals into a composite NTSC signal and keys 10/20k graphics over an external video source.

1) MIX VID OUT. This output is the 10/20k graphics keyed over the KEY BKGD IN. During the blanking interval (which includes front porch, sync, breezeway, burst and back porch), the mixed video output consists of the signal supplied by the keyer background input. In other words, the sync is stripped off the background input and is used in the output.

2) KEY BKGD IN. This input is the video signal which the 10/20k graphics will be keyed over.

3) KEY IN. Comes from KEY OUT on the display card.

4) PGM OUT. Two (2) outputs have this label. One goes to PGM IN on the display card. The other is sent to the switcher, tape machine or wherever. They both are NYSC outputs. The output which is a coaxicon connector should go to the display card.

5) PGM OUT. Explained above.

6) RED OUT. This is a 0.707 volt signal with set-up digitally added.

7) GREEN OUT. This is a 0.707 volt signal.

TIMING ADJUSTMENTS

The following measurements are made at the output of the switcher by selecting between the reference source and the 10/20k being adjusted on the switcher. An externally locked waveform monitor and vectorscope should be connected to the output of the switcher. The house reference must be connected to the reference video input of the 10/20k. All the timing controls for the 10/20k are accessible from the rear of the machine. It may be helpful to consult the manual for your switcher to perform these adjustments.

1) Adjust the fine phase control until it is completely counter-clockwise. The coarse phase adjustment varies the subcarrier phase in 90 degree steps. Adjust the coarse phase switch (SW1) until the burst phase of the reference and the 10/20k are within 90 degrees of each other.

2) Adjust the fine phase potentiometer (VR1) until the subcarrier phase of the reference and the 10/20k are identical. It is preferable to have the fine phase control in the first 90 degrees of its travel from counter-clockwise to clockwise.

3) The horizontal timing may be adjusted over a 1/2 scan line (approximately 32 uS) interval by adjusting the coarse sync switch (SW1). Adjust the coarse sync switch until the leading edge of the reference and 10/20k sync pulses are within a few microseconds of each other.

4) Use the fine sync switch (SW2) until the sync pulses are within 280 nS of each other. It may be necessary to readjust the coarse sync control to accomplish this.

5) The extra fine sync delay line varies the horizontal timing over a range of 120 nS. Adjust the extra fine sync delay line (DL1) until the reference and 10/20k are horizontally timed within 10 nS. **CAUTION:** This adjustment is very delicate and may be damaged if the shaft is forced beyond the stop. If the reference video being sent to the 10/20k is correctly SC/H phased, the 10/20k should also be correctly SC/H phased. This may be verified with an SC/H phase meter.

ENCODER SECTION ADJUSTMENTS

1) Adjust VR14 for the proper sync level (0 IRE to -40 IRE) appearing at the program output P12. Note: this actually adjusts the entire video level.

2) Set the waveform monitor for 1 uS per division. Adjust VRB for minimal residual subcarrier in the sync portion of the signal appearing at the program output P12.

3) Adjust VR11 for minimal residual subcarrier in the sync portion of the signal appearing at the program output P12. Repeat steps 2 and 3 as many times as necessary to reduce the residual subcarrier at the output to a minimum.

4) Set the waveform monitor for 2H. Connect the red output (P11) to the input of the waveform monitor. Adjust VR2 to obtain full red at 100 IRE and blanking at 0 IRE. There is also a set-up and a 75% level that appears on this signal and on the green and the blue outputs which should be at approximately 7.5 IRE and 75 IRE, respectively.

5) Connect the green output (P10) to the input of the waveform monitor. Adjust VR4 to obtain full green at 100 IRE and blanking at 0 IRE.

6) Connect the blue output (P9) to the input of the waveform monitor. Adjust VR9 to obtain full blue at 100 IRE and blanking at 0 IRE.

7) Connect the program output (P12) to the input of the waveform monitor. Adjust VR3 (red), VR5 (green) and VR7 (blue) to obtain full white at 100 IRE with a minimal amount of residual subcarrier. If the white level is off by a large amount, you may wish to use an oscilloscope to adjust pins 3, 4 and 5 of the MC1377 (U21) for a level of 1.2 volts from blanking to full color level at each pin.

8) Adjust the quadrature by adjusting VR9 and viewing the vectors on the vectorscope.

9) Adjust VR10 so the trace from the green to magenta crosses through the origin. Note: this also alters the chroma level.

10) Adjust the potentiometer by P14 for the correct level of chroma by viewing the burst vector on the vectorscope. The vector should extend from the origin to the 75% mark on the vectorscope. Repeat steps 7 through 10 as many times as necessary.

11) Set the waveform monitor for 1 uS. Adjust VR12 until the zero-crossing (positive or negative slope) that precedes the first half cycle of subcarrier that is 50% or greater of the burst amplitude, occurs 0.6 uS after the rising edge of sync.

12) Adjust VR13 for 8 to 11 cycles of subcarrier, or 2.5 uS of color burst.

LINEAR KEYER ADJUSTMENTS

All timing information (sync and subcarrier) for the linear keyer section is provided by the background signal. So, as a general rule, tape machines without a time base corrector is unacceptable as a background signal. For the following adjustments, it is necessary to connect a stable background, such as house color bars, to the background input of the 10/20k. It is not absolutely necessary to have the source of color bars timed into the system for the initial adjustments.

1.) Set-up the 10/20k to produce a gray scale. This is accomplished by the following key strokes; hit the MISC key, select the character for diagnostics, select the character for gray scales, and hit the space bar 3 times. The 10/20k should now be producing a smoothly increasing gray scale with black on the left and full white on the right of the screen. The key signal associated with this picture is also a gray scale.

2) Connect an oscilloscope to the point labeled TP1 on the SEK card. The stiffening bar on the card is a convenient ground point. The signal on the oscilloscope should look something like a sawtooth waveform with sync added to it. It may be desirable to reduce the oscilloscope's bandwidth to 20 MHz.

3) Adjust VR16 until the blanking level associated with the signal on the oscilloscope is at 0.0 volts.

4) Adjust VR18 until the white level associated with the signal on the oscilloscope is at 1.0 volts.

5) Connect a calibrated source of color bars for the background input of the 10/20k.

6) Connect a waveform monitor to the mixed video output of the 10/20k. Adjust the waveform monitor to display 2 scan lines worth of video information. The combination of the gray, color bars and key signal should be producing a signal which looks like color bars on the left and gradually smearing into white at the right of the screen.

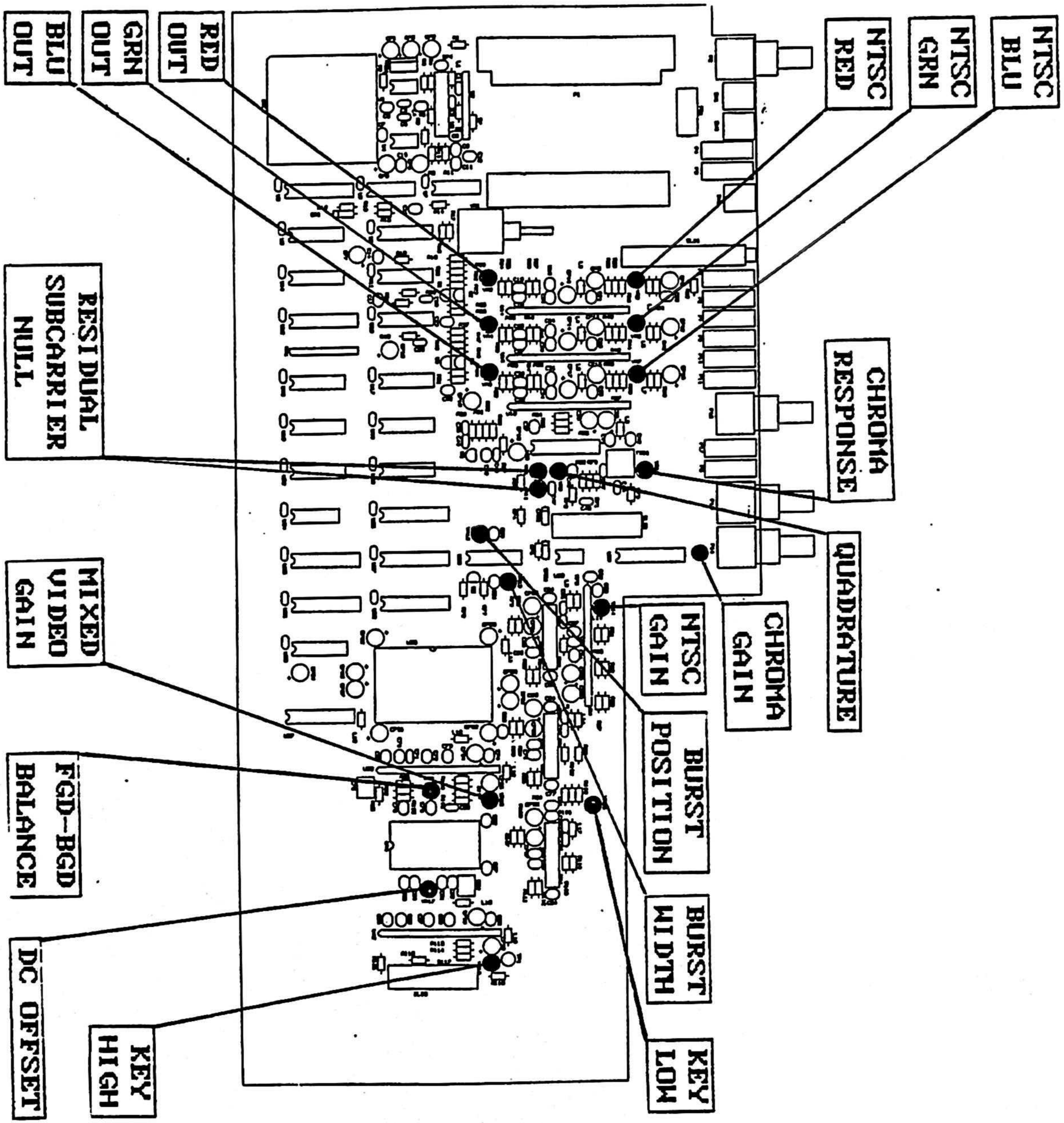
7) View the waveform monitor R99 until the white level from the color bars and the white level from the gray scale are the same level. Do not worry about the absolute levels of the whites at this point.

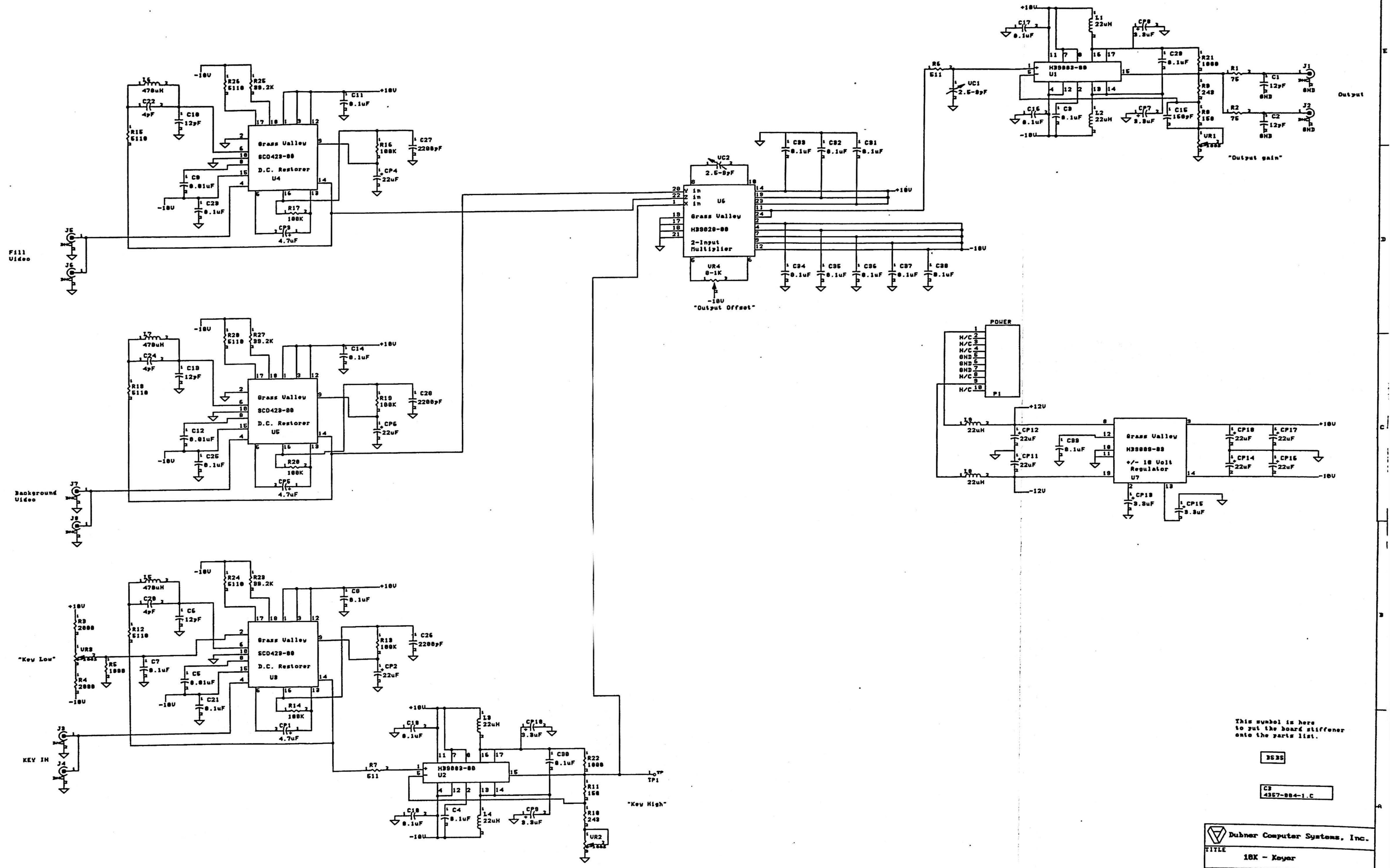
8) Adjust VR15 until the signal on the waveform monitor is the correct amplitude (both whites at 100 IRE and sync at -40 IRE). It may be necessary to repeat the step again to make all the levels the correct amplitude.

9) To adjust the level of subcarrier, look at the vectorscope and adjust C93 for the correct level of burst. It may be necessary to also adjust C76 to obtain the full amount of color burst required.

10) Connect the oscilloscope to P14, the mixed video output. Adjust VR17 until the blanking level is at ground potential.

11) To verify that the linear keyer section is adjusted properly, perform the following keystrokes; hit the CONFIG key, select the character to turn the keyer on (graphics only). All that should now appear on the waveform monitor is the gray scale. If there is any "bleed through" of the color bars into the gray scale, the keyer is not adjusted properly. Performing steps 3 and 4 again while also viewing the waveform monitor should correct any "bleed through" problems.


NTSC BLU
NTSC GRN
NTSC RED
RED OUT
GRN OUT
BLU OUT
CHROMA RESPONSE
RESIDUAL SUBCARRIER NULL
QUADRATURE
CHROMA GAIN
NTSC GAIN
MIXED VIDEO GAIN
BURST POSITION
FGD-BGD BALANCE
BURST WIDTH
DC OFFSET
KEY HIGH
KEY LOW


Fill Video
J5
J6
Background Video
J7
J8
KEY IN
J3
J4
"Key Low"
"Key High"
TP1
Grass Valley
SC0429-00
D.C. Restorer
U4
U5
U3
Grass Valley
H39020-00
2-Input Multiplier
U6
"Output Offset"
VR4 0-1K
VC2 2.5-9pF
H39003-00
U1
U2
VC1 2.5-9pF
"Output gain"
Output
J1
J2
POWER
P1
N/C
GND
Grass Valley
H39008-00
+/- 10 Volt Regulator
U7
+10V
-10V
+12V
-12V

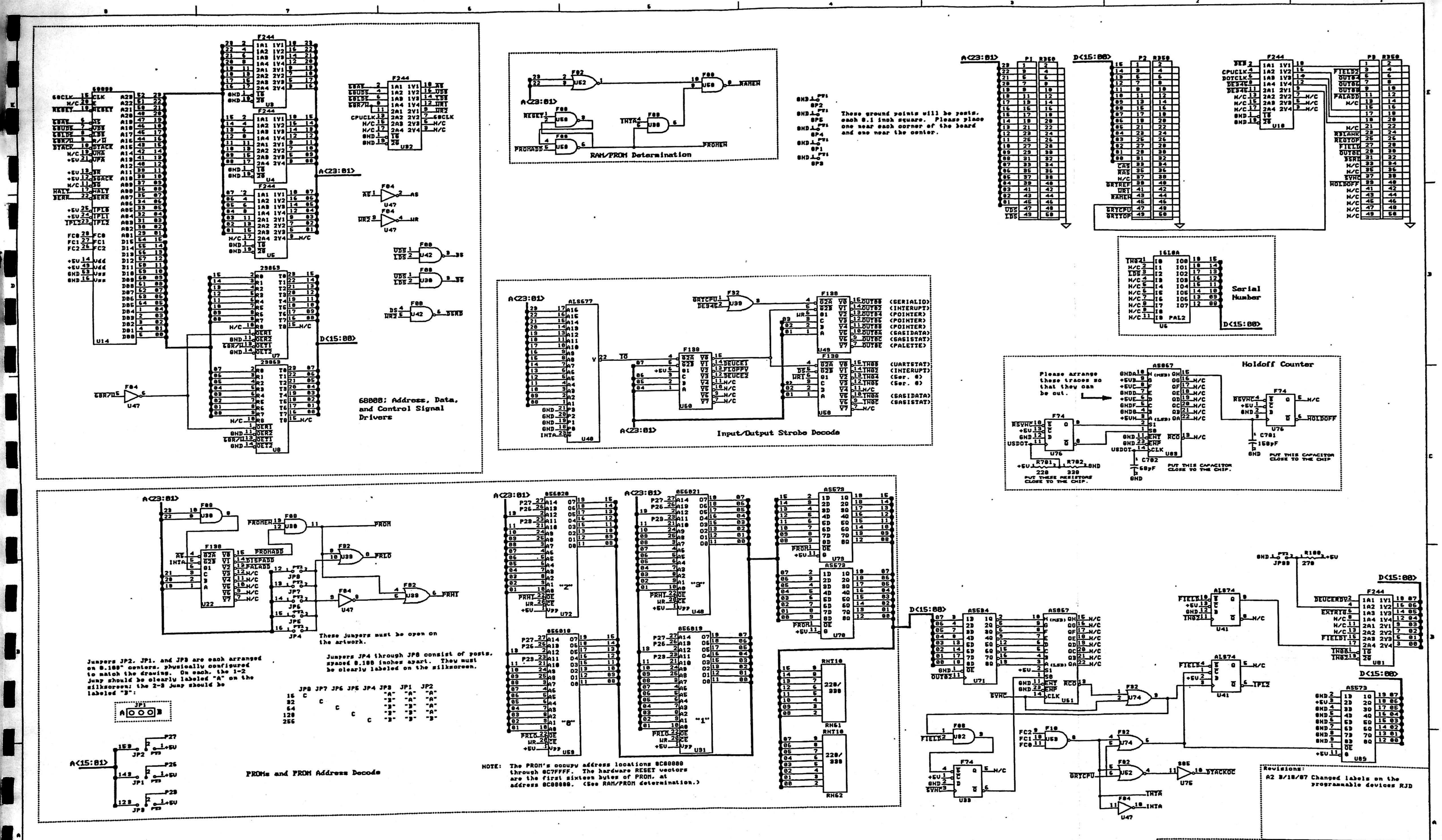
This symbol is here
to put the board stiffener
onto the parts list.

3535

C3
4357-004-1.C

Dubner Computer Systems, Inc.

TITLE			
10K - Keyer			
SIZE D	REV C 10/85	NUMBER 4357-004-1	BY RJD 10/84
DATE	26 Aug 1986	SHEET	1 of 2



68000: Address, Data, and Control Signal Drivers
RAM/PROM Determination
Input/Output Strobe Decode
Serial Number
Holdoff Counter
PROMs and PROM Address Decode
Please arrange these traces so that they can be cut.
PUT THIS CAPACITOR CLOSE TO THE CHIP
PUT THESE RESISTORS CLOSE TO THE CHIP.
A<23:01>
D<15:00>
A<15:01>

These ground points will be posts, each 0.1 inch square. Please place one near each corner of the board and one near the center.

These jumpers must be open on the artwork.

Jumpers JP2, JP1, and JP3 are each arranged on 0.100" centers, physically configured to match the drawing. On each, the 1-2 jump should be clearly labeled "A" on the silkscreen; the 2-3 jump should be labeled "B":

Jumpers JP4 through JP8 consist of posts, spaced 0.100 inches apart. They must be clearly labeled on the silkscreen.

	JP8	JP7	JP6	JP5	JP4	JP3	JP1	JP2
						"A"	"A"	"A"
16	C					"B"	"A"	"A"
32		C				"B"	"A"	"A"
64			C			"B"	"B"	"A"
128				C		"B"	"B"	"A"
256					C	"B"	"B"	"B"

NOTE: The PROM's occupy address locations 0C00000 through 0C7FFFF. The hardware RESET vectors are the first sixteen bytes of PROM, at address 0C00000. (See RAM/PROM determination.)

Revisions:

A2 3/18/87 Changed labels on the programmable devices RJD

Latest Drawing Revs:

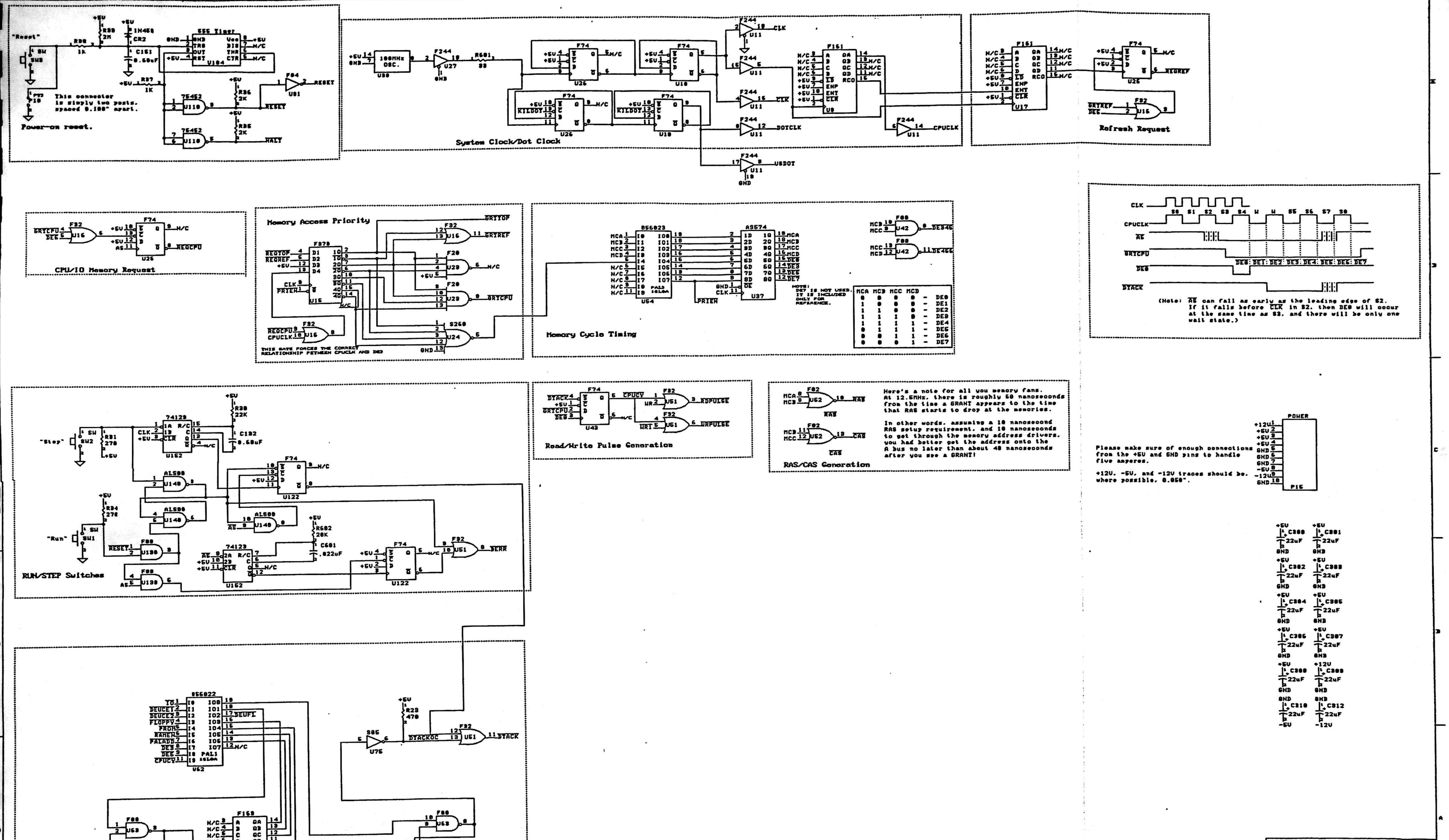
Sheet 1: A2
Sheet 2: A2
Sheet 3: A2
Sheet 4: A2
Sheet 5: A2

Dubner Computer Systems, Inc.

TITLE 10K - CPU

SIZE	REV	NUMBER	BY
D	A2	040010-00	RJD 10/84

DATE 02 Feb 1987 SHEET 1 of 5


Power-on reset.
System Clock/Dot Clock
Refresh Request
CPU/IO Memory Request
Memory Access Priority
THIS GATE FORCES THE CORRECT RELATIONSHIP BETWEEN CPUCLK AND DE0
Memory Cycle Timing
RUN/STEP Switches
Read/Write Pulse Generation
RAS/CAS Generation
DTACK Generation
"Reset"
"Step"
"Run"
This connector is simply two posts, spaced 0.100" apart.
555 Timer
100MHz OSC.
CLK
CPUCLK
DOTCLK
USDOT
RESET
HALT
GRTREF
GRTCPU
GRTIOP
REQCPU
REQREF
PRIEN
BERR
DTACK
RDPULSE
WRPULSE
RAS
CAS
POWER
P15
CLK
CPUCLK
AS
GRTCPU
DE0
DTACK
S0 S1 S2 S3 S4 W W S5 S6 S7 S0
DE0 DE1 DE2 DE3 DE4 DE5 DE6 DE7

NOTE:
DE7 IS NOT USED.
IT IS INCLUDED
ONLY FOR
REFERENCE.

MCA	MCB	MCC	MCD		
0	0	0	0	-	DE0
1	0	0	0	-	DE1
1	1	0	0	-	DE2
1	1	1	0	-	DE3
1	1	1	1	-	DE4
0	1	1	1	-	DE5
0	0	1	1	-	DE6
0	0	0	1	-	DE7

Here's a note for all you memory fans. At 12.5MHz, there is roughly 50 nanoseconds from the time a GRANT appears to the time that RAS starts to drop at the memories.

In other words, assuming a 10 nanosecond RAS setup requirement, and 10 nanoseconds to get through the memory address drivers, you had better get the address onto the A bus no later than about 40 nanoseconds after you see a GRANT!

(Note: $\overline{\text{AS}}$ can fall as early as the leading edge of S2. If it falls before $\overline{\text{CLK}}$ in S2, then DE0 will occur at the same time as S3, and there will be only one wait state.)

Please make sure of enough connections from the +5V and GND pins to handle five amperes.

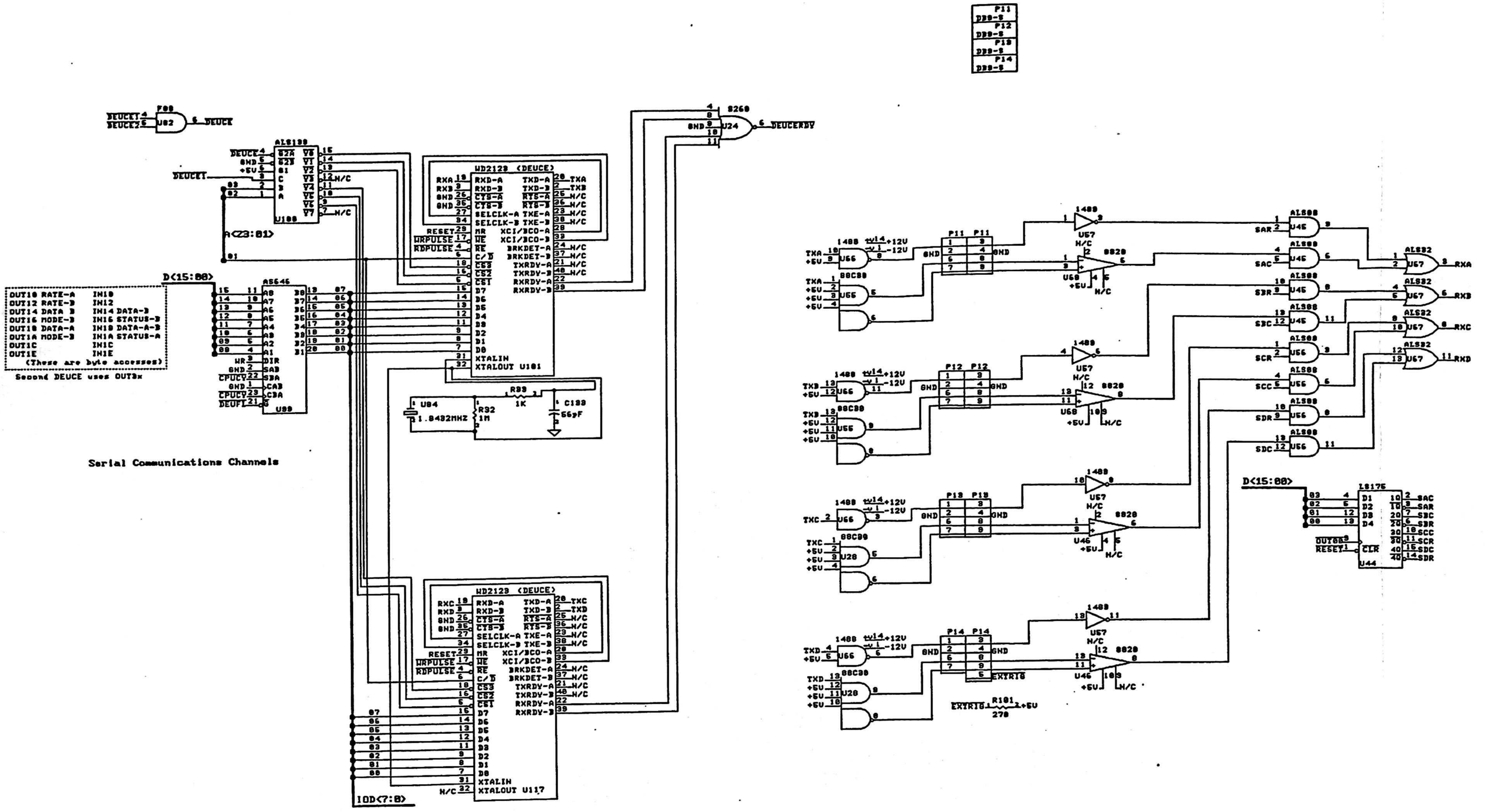
+12V, -5V, and -12V traces should be, where possible, 0.050".

Dubner Computer Systems, Inc.

TITLE 10K - CPU

SIZE	REV	NUMBER	BY
D	A2	040010-00	RJD 10/84

DATE 02 Feb 1987 SHEET 2 of 5



F00
DEUCE1
DEUCE2
U82
DEUCE
ALS138
DEUCE
GND
+5V
G2A
G2B
G1
C
B
A
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
U100
H/C
A<23:01>
8260
GND
U24
DEUCERDY
WD2123 (DEUCE)
RXA
RXB
GND
RXD-A
RXD-B
CTS-A
CTS-B
SELCLK-A
SELCLK-B
MR
WE
RE
C/D
CS3
CS2
CS1
D7
D6
D5
D4
D3
D2
D1
D0
XTALIN
XTALOUT
U101
TXD-A
TXD-B
RTS-A
RTS-B
TXE-A
TXE-B
XCI/BCO-A
XCI/BCO-B
BRKDET-A
BRKDET-B
TXRDY-A
TXRDY-B
RXRDY-A
RXRDY-B
TXA
TXB
RESET
WRPULSE
RDPULSE
D<15:00>
AS646
WR
DIR
SAB
SBA
CAB
CBA
CPUCY
DEUFI
U99
U84
1.8432MHZ
R32
1M
R33
1K
C133
56pF
RXC
RXD
TXC
TXD
U117
IOD<7:0>
P11
DB9-S
P12
P13
P14
1488
1489
U66
U67
U68
U46
U28
88C30
8820
ALS08
ALS32
U45
U56
U44
LS175
D1
D2
D3
D4
CLR
OUT08
+12V
-12V
+5V
EXTRIG
R101
270
SAR
SAC
SBR
SBC
SCR
SCC
SDR
SDC

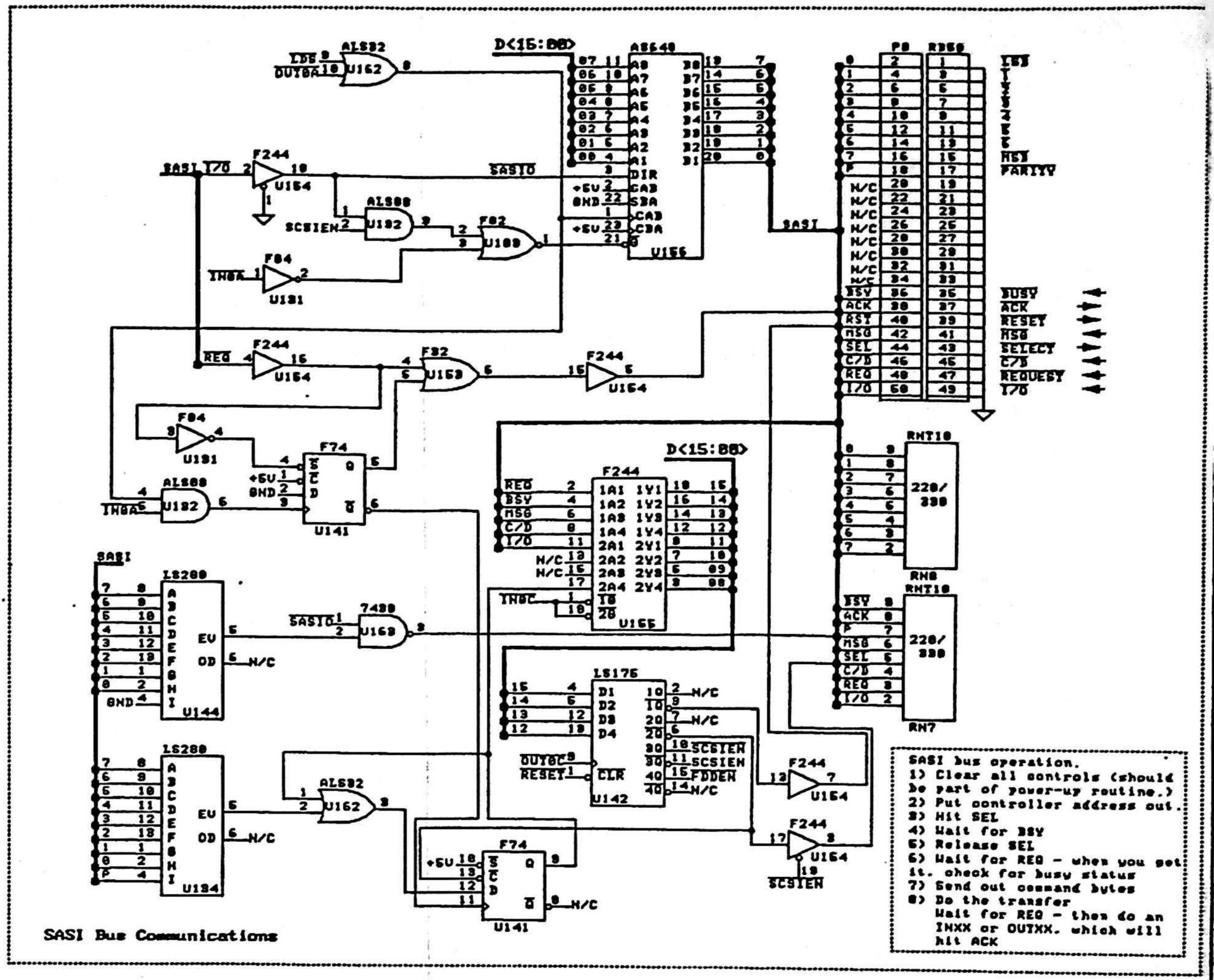
OUT10 RATE-A   IN10
OUT12 RATE-B   IN12
OUT14 DATA B   IN14 DATA-B
OUT16 MODE-B   IN16 STATUS-B
OUT18 DATA-A   IN18 DATA-A-B
OUT1A MODE-B   IN1A STATUS-A
OUT1C   IN1C
OUT1E   IN1E
(These are byte accesses)

Second DEUCE uses OUT3x

Serial Communications Channels

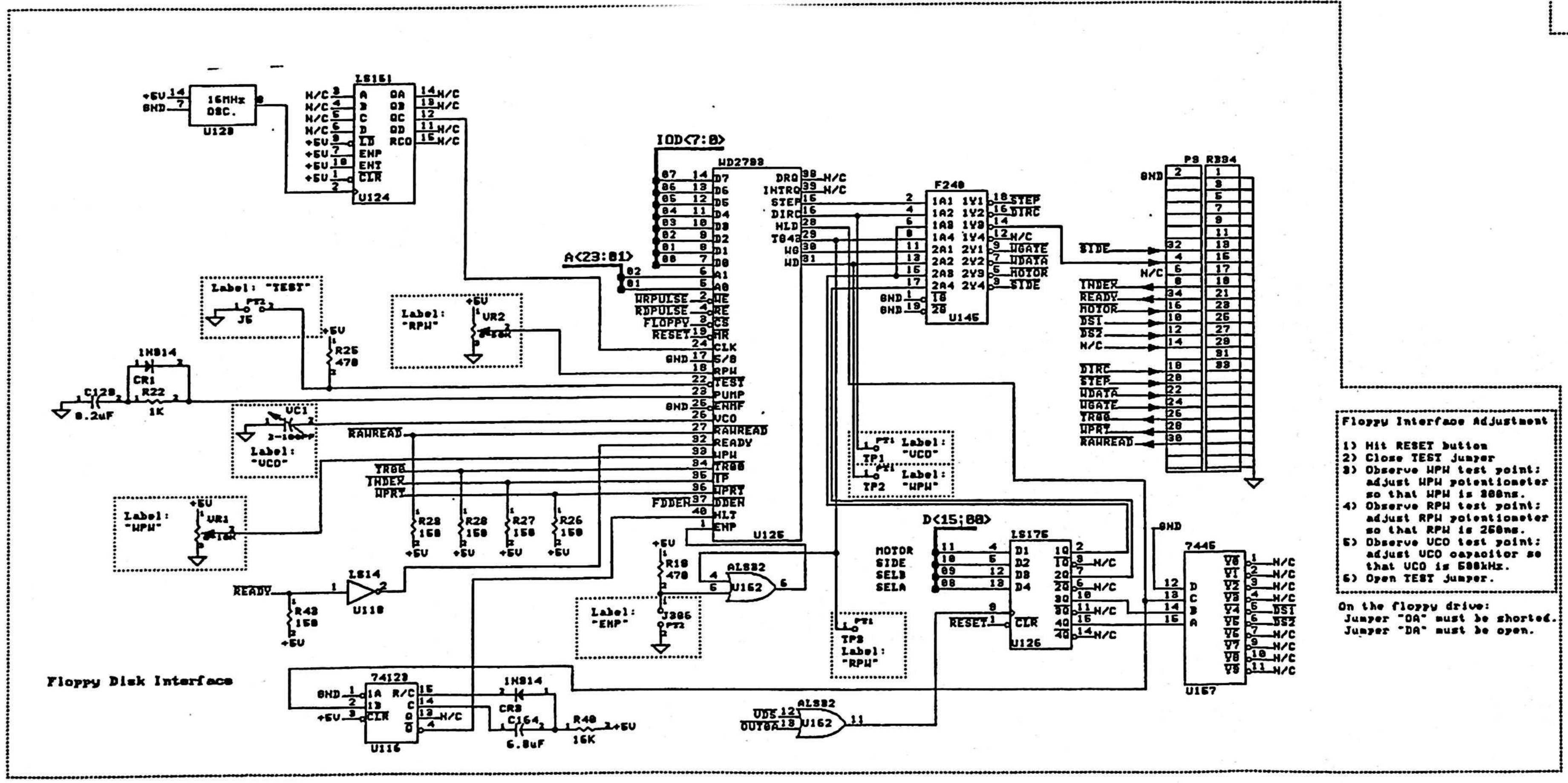
Dubner Computer Systems, Inc.

TITLE			
10K - CPU			
SIZE	REV	NUMBER	BY
D	A2	040010-00	RJB 10/84
DATE 02 Feb 1987		SHEET 3 of 5	


D<15:00>
AS640
U155
ALS32
U162
F244
U154
SASI I/O
SASIO
ALS08
U192
SCSIEN
F02
U189
F04
U131
INSA
SASI
P8
R350
I03
MSI
PARITY
BSY
ACK
RST
MSG
SEL
C/D
REQ
I/O
BUSY
ACK
RESET
MSG
SELECT
C/D
REQUEST
I/O
U163
F74
U141
INSA5
D<15:00>
F244
U155
RNT10
220/
330
RN8
RNT10
220/
330
RN7
LS280
U144
U134
SASIO
74S00
U169
EV
OD
N/C
INSC
LS175
U142
OUTSC
RESET
CLR
SCSIEN
FDDEN
U154
SCSIEN
+5V
GND
N/C

SASI Bus Communications

SASI bus operation.
1) Clear all controls (should be part of power-up routine.)
2) Put controller address out.
3) Hit SEL
4) Wait for BSY
5) Release SEL
6) Wait for REQ - when you get it. check for busy status
7) Send out command bytes
8) Do the transfer
Wait for REQ - then do an INXX or OUTXX. which will hit ACK


+5V
GND
16MHz
OSC.
U129
LS161
U124
N/C
IOD<7:0>
WD2793
U125
DRQ
INTRQ
STEP
DIRC
HLD
TG43
WG
WD
A<23:01>
WRPULSE
RDPULSE
FLOPPY
RESET
CLK
5/8
RPW
TEST
PUMP
ENMF
VCO
RAWREAD
READY
WPW
TR00
IP
WPRT
DDEN
HLT
ENP
Label: "TEST"
J5
R25
470
Label: "RPW"
VR2
1N914
CR1
R22
1K
C128
8.2uF
VC1
Label: "VCO"
Label: "WPW"
VR1
R28
150
R27
R26
R43
READY
LS14
U118
TR00
INDEX
WPRT
FDDEN
+5V
R18
470
ALS32
U162
Label: "ENP"
J385
TP1
Label: "VCO"
TP2
Label: "WPW"
TP3
Label: "RPW"
F240
U145
STEP
DIRC
N/C
WGATE
WDATA
MOTOR
SIDE
D<15:00>
LS175
U126
MOTOR
SIDE
SEL3
SELA
RESET
CLR
7445
U157
DS1
DS2
P5
R394
GND
SIDE
INDEX
READY
MOTOR
DS1
DS2
DIRC
STEP
WDATA
WGATE
TR00
WPRT
RAWREAD
74123
U116
1N914
CR3
C164
6.8uF
R48
16K
UDK
OUTDR
ALS32
U162

Floppy Disk Interface

Floppy Interface Adjustment

1) Hit RESET button
2) Close TEST jumper
3) Observe WPW test point: adjust WPW potentiometer so that WPW is 300ns.
4) Observe RPW test point: adjust RPW potentiometer so that RPW is 250ns.
5) Observe VCO test point: adjust VCO capacitor so that VCO is 500kHz.
6) Open TEST jumper.

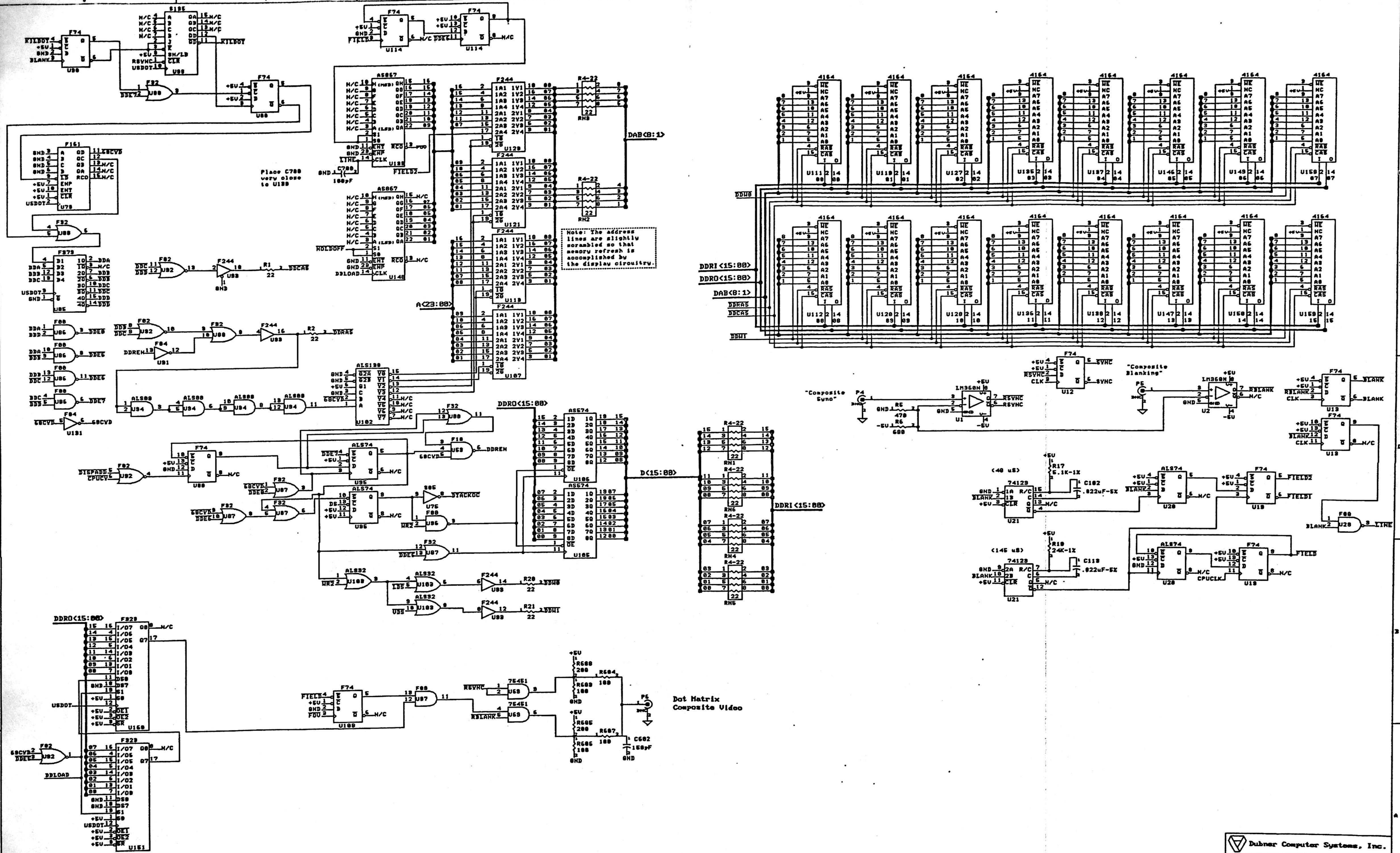
On the floppy drive:
Jumper "DA" must be shorted.
Jumper "DA" must be open.

Dubner Computer Systems, Inc.

TITLE
10X - CPU

SIZE	REV	NUMBER	BY
D	AZ	040010-00	RJD 10/84

DATE 02 Feb 1987
SHEET 4 of 5

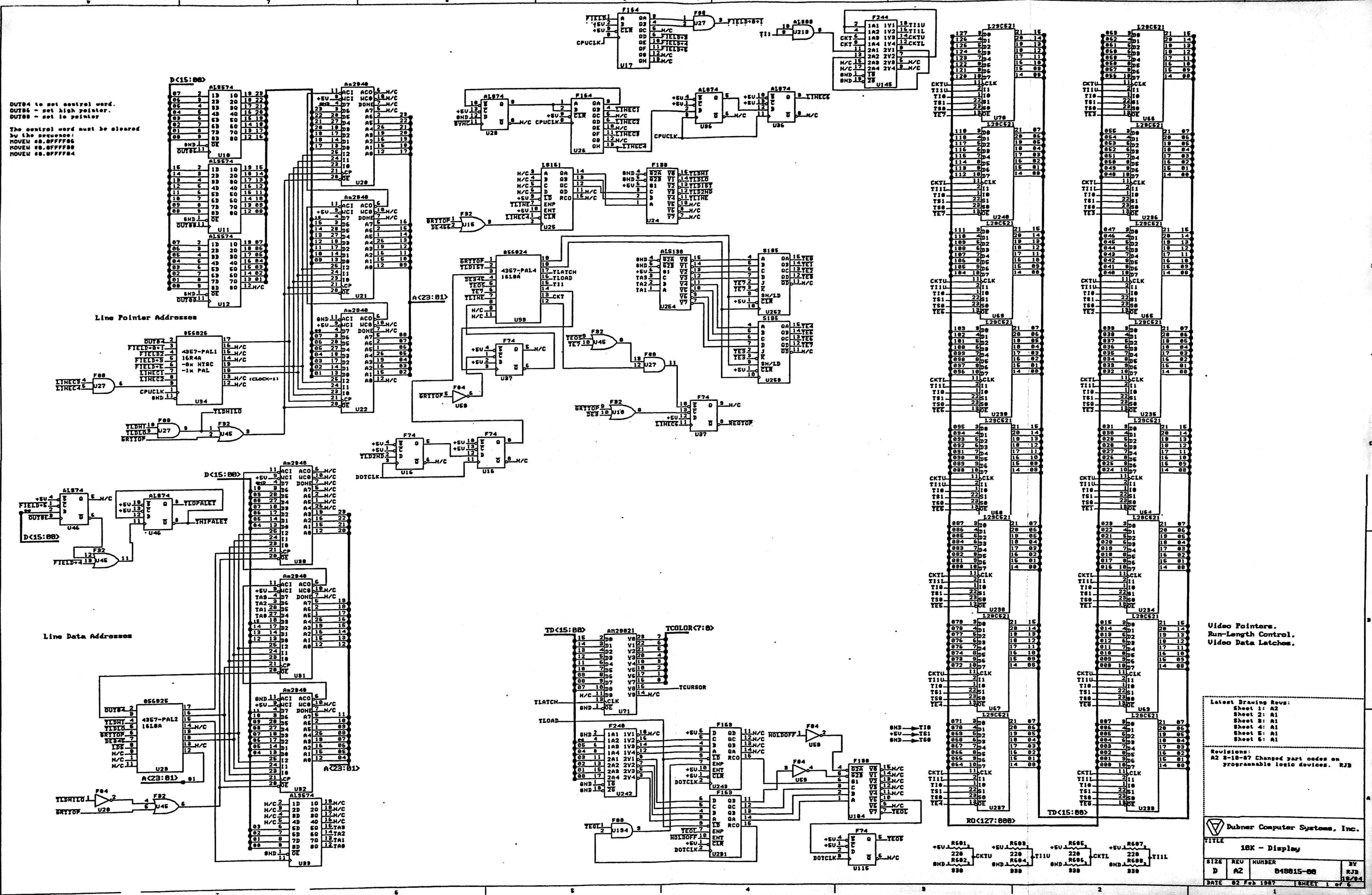

Place C700 very close to U130
Note: The address lines are slightly scrambled so that memory refresh is accomplished by the display circuitry.
DAB<8:1>
A<23:00>
DDRO<15:00>
DDRI<15:00>
D<15:00>
DDWE
DDHI
DDRAS
DDCAS
4164
F244
AS574
AS867
ALS138
F74
F161
F373
8195
R4-22
"Composite Sync"
"Composite Blanking"
LM360N
74123
(40 uS)
(145 uS)
F929
75451
Dot Matrix Composite Video
FIELD
FIELD1
FIELD2
LINE
BLANK
SYNC
RSYNC
RBLANK
STACKOC
DDREN
KILDOT
USDOT
DBLOAD
HOLDOFF
68CVD
DISPADR
CPUCLK

Dubner Computer Systems, Inc.

TITLE 10K - CPU

SIZE D | REV A2 | NUMBER 048010-00 | BY RJD 10/84

DATE 02 Feb 1987 | SHEET 5 of 5



D<15:00>
ALS574
U10
U11
U12
Am2940
U20
U21
U22
A<23:01>
Line Pointer Addresses
056826
4357-PAL1
16R4A
-8x NTSC
-1x PAL
U34
F164
U17
F86
U27
ALS00
U219
F244
U145
ALS74
U29
U26
U36
LS161
U25
F138
U24
F32
U15
056824
4357-PAL4
16L8A
U98
ALS138
U254
S195
U252
U259
F74
U37
F04
U59
U16
U46
Line Data Addresses
U38
U31
U32
056825
4357-PAL2
16L8A
U23
U33
TD<15:00>
AM29821
TCOLOR<7:0>
U71
F240
U242
F169
U249
U281
U194
U184
U116
L29C521
U78
U248
U69
U238
U68
U298
U67
U237
U66
U236
U65
U235
U64
U234
U63
U233
RD<127:000>
R601
R602
R603
R604
R605
R606
R607
R608
220
330
CKTU
TI1U
CKTL
TI1L

OUT04 to set control word.
OUT06 - set high pointer.
OUT00 - set lo pointer

The control word must be cleared
by the sequence:
MOVEW $0.0FFFF06
MOVEW $0.0FFFF00
MOVEW $0.0FFFF04

Video Pointers.
Run-Length Control.
Video Data Latches.

Latest Drawing Revs:
Sheet 1: A2
Sheet 2: A1
Sheet 3: A1
Sheet 4: A1
Sheet 5: A1
Sheet 6: A1

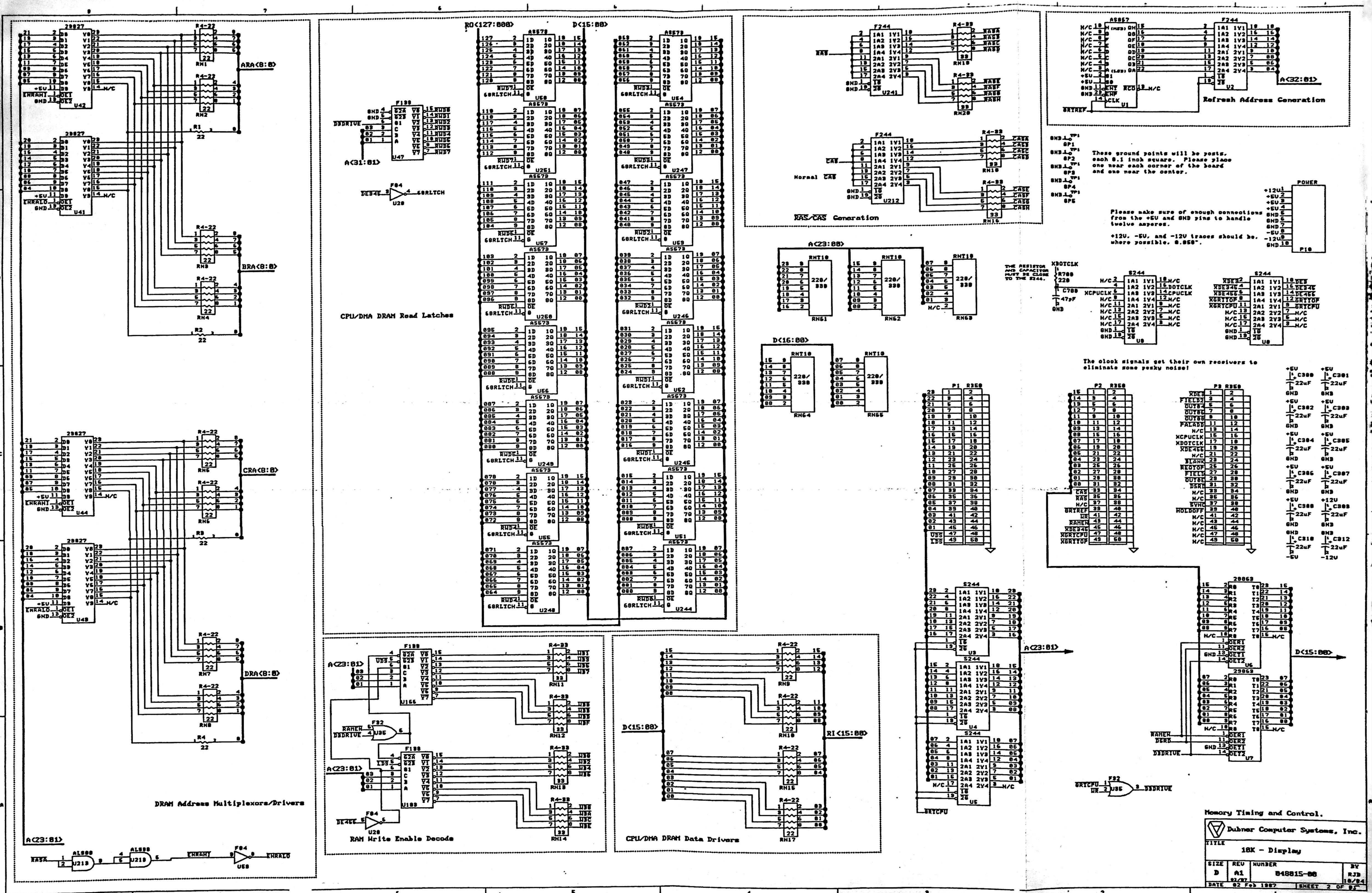
Revisions:
A2 3-10-87 Changed part codes on
programmable logic devices. RJD

Dubner Computer Systems, Inc.

TITLE
10K - Display

SIZE	REV	NUMBER	BY
D	A2	040015-00	RJD 10/84

DATE 02 Feb 1987 SHEET 1 of 6


DRAM Address Multiplexors/Drivers
CPU/DMA DRAM Read Latches
RAM Write Enable Decode
CPU/DMA DRAM Data Drivers
RAS/CAS Generation
Normal CAS
Refresh Address Generation
POWER
ARA<8:0>
BRA<8:0>
CRA<8:0>
DRA<8:0>
A<23:01>
RD<127:000>
D<15:00>
RI<15:00>
A<31:01>
A<32:01>
D<15:00>
ENRAHI
ENRALO
68RLTCH
DBDRIVE
RAMEN
XDOTCLK
DOTCLK
XCPUCLK
CPUCLK
BRTREF
BRTCPU
P1
P2
P3
P10
U1
U2
U3
U4
U5
U6
U7
U8
U9

These ground points will be posts, each 0.1 inch square. Please place one near each corner of the board and one near the center.

Please make sure of enough connections from the +5V and GND pins to handle twelve amperes.

+12V, -5V, and -12V traces should be, where possible, 0.050".

THE RESISTOR AND CAPACITOR MUST BE CLOSE TO THE S244.

The clock signals get their own receivers to eliminate some pesky noise!

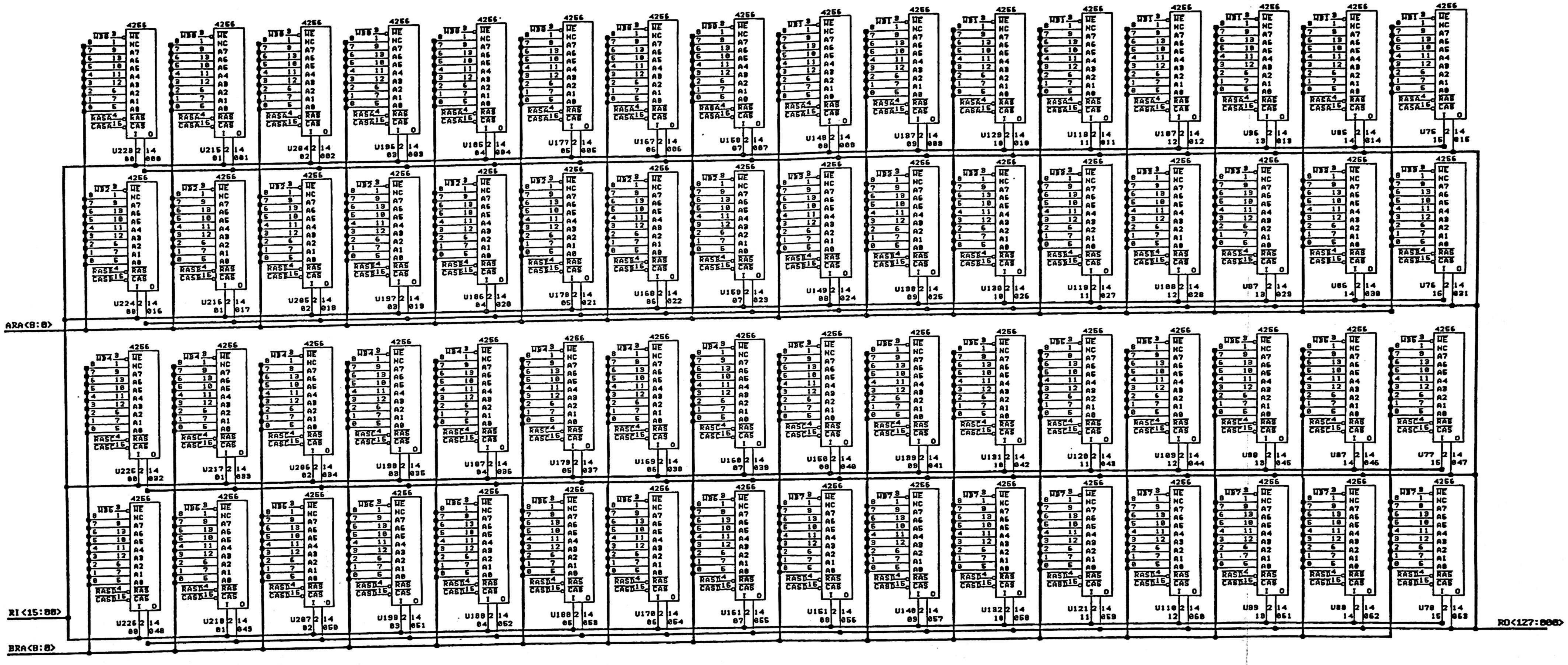
Memory Timing and Control.

Dubner Computer Systems, Inc.

TITLE
10K - Display

SIZE	REV	NUMBER	BY
D	A1 02/87	040015-00	RJD 10/84

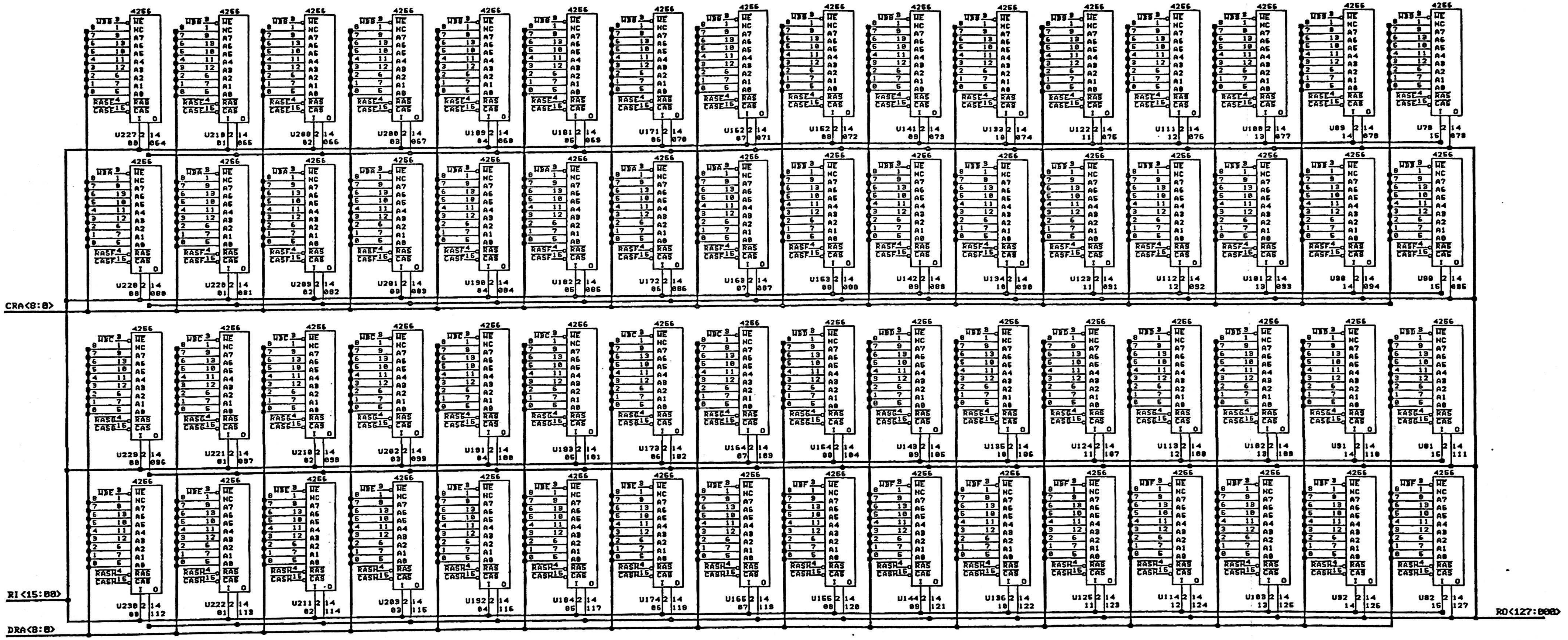
DATE 02 Feb 1987 SHEET 2 OF 05


ARA<8:0>
RI<15:00>
BRA<8:0>
RO<127:000>
4266

Memory Bytes 00 through 07.

Dubner Computer Systems, Inc.

TITLE	10K - Display		
SIZE D	REV A1 02/87	NUMBER 040015-00	BY RJB 10/84
DATE 12 Sept 1986		SHEET 3 OF 8	


CRA<8:0>
RI<15:00>
DRA<8:0>
RO<127:000>
4256
WE
NC
A7
A6
A5
A4
A3
A2
A1
A0
RAS
CAS
I
O
WBA
WBB
WBC
WBD
WBE
WBF
RASE
CASE
RASF
CASF
RASG
CASG
RASH
CASH
U227
U219
U208
U200
U228
U220
U209
U201

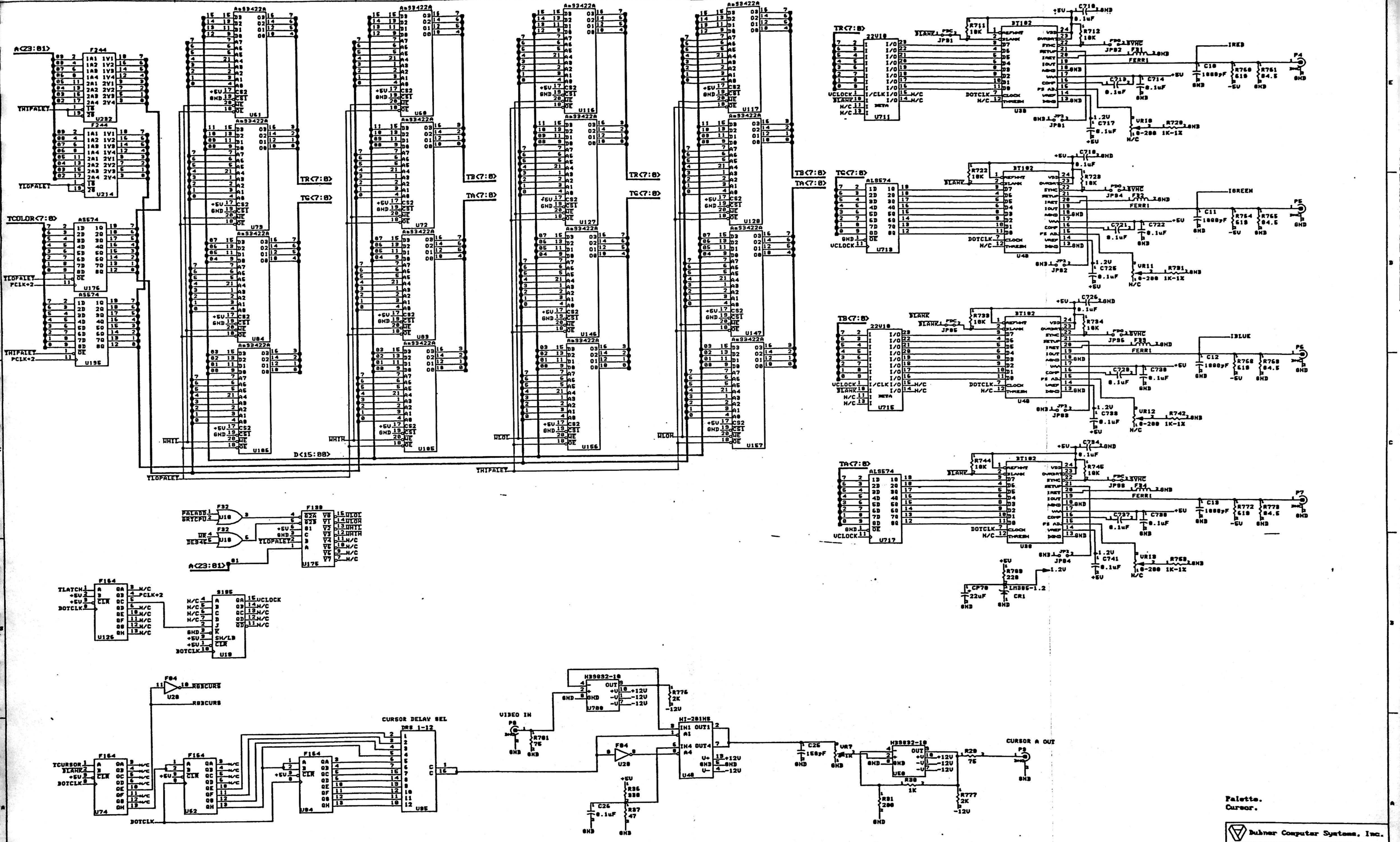
Memory Bytes 08 through 0F.

Dubner Computer Systems, Inc.

TITLE
18K - Display

SIZE	REV	NUMBER	BY
D	A1 01/07	040015-00	RJB

DATE 12 Sept 1986 SHEET 4 OF 05


A<23:01>
TCOLOR<7:0>
TLOPALET
THIPALET
PCLK+2
F244
AS574
Am93422A
D<15:00>
TR<7:0>
TG<7:0>
TB<7:0>
TA<7:0>
WHIT
WLOT
WHIH
WLOH
F32
F138
PALADD
BRYCPU
DEB4R
F164
S195
TLATCH
DOTCLK
VCLOCK
22V10
ALS574
BT102
BLANK
IRED
IGREEN
IBLUE
FERRI
C10
C11
C12
C13
P4
P5
P6
P7
LM385-1.2
1.2V
TCURSOR
CURSOR DELAY SEL
DRS 1-12
VIDEO IN
P8
HA5032-10
HI-201HS
CURSOR A OUT
P9
R00CURS
R01CURS

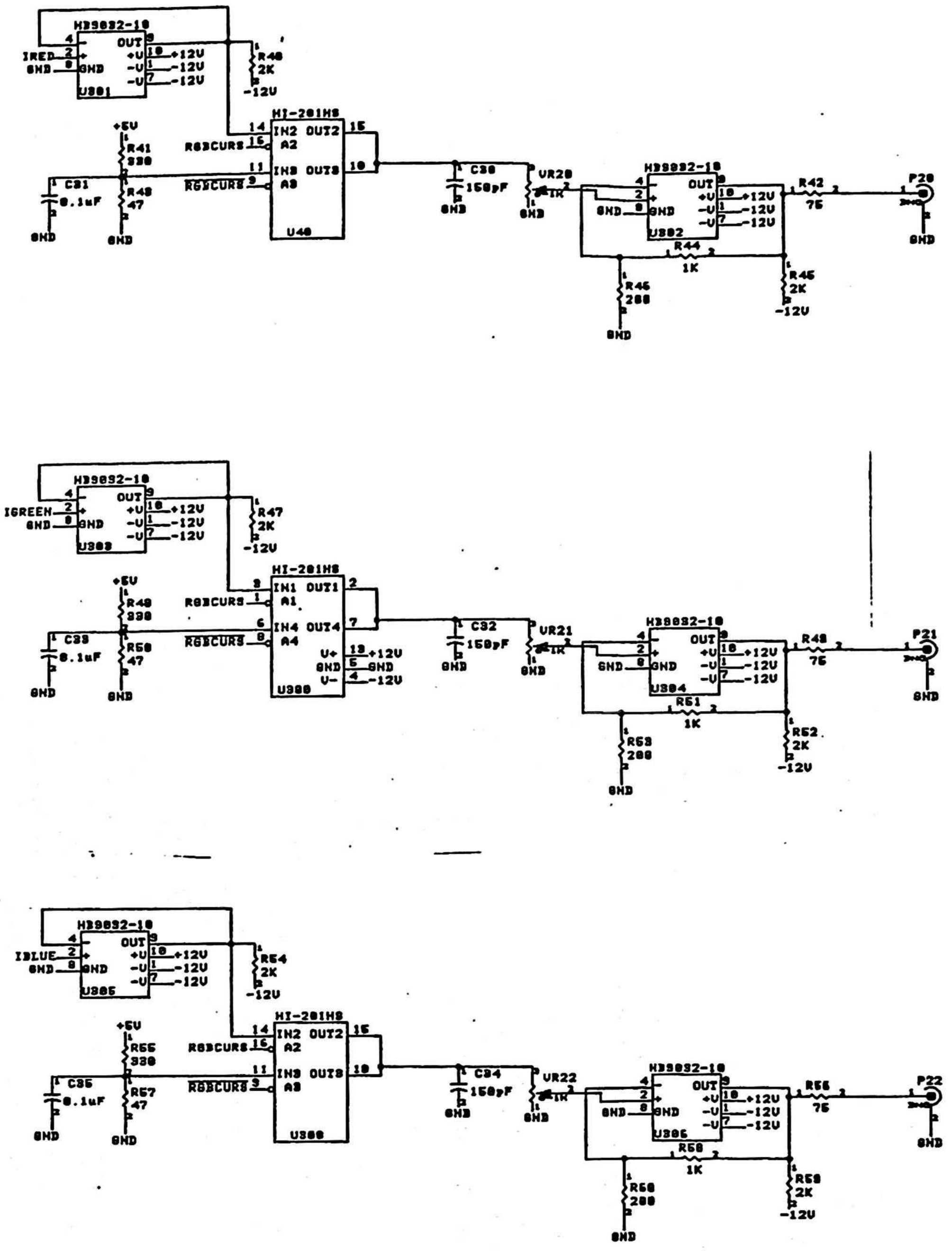
Palette.
Cursor.

Dubner Computer Systems, Inc.

TITLE
16K - Display Plane

SIZE	REV	NUMBER	BY
D	A1	040015-00	RJ3 10/84

DATE 02 Feb 1987
SHEET 5 of 6


HA5002-10
IRED
GND
OUT
+U
-U
+12V
-12V
U301
R40
2K
-12V
HI-201HS
IN2
OUT2
A2
IN3
OUT3
A3
RGBCURS
U40
+5V
R41
330
R42
47
C31
0.1uF
C30
150pF
VR20
1K
GND
R42
75
P20
R44
1K
R46
200
R45
2K
U302
IGREEN
U303
R47
2K
IN1
OUT1
A1
IN4
OUT4
A4
V+
V-
U300
R48
330
R50
47
C33
0.1uF
C32
150pF
VR21
R49
75
P21
R51
1K
R53
200
R52
2K
U304
IBLUE
U305
R54
2K
R55
330
R57
47
C35
0.1uF
C34
150pF
VR22
R56
75
P22
R58
1K
R60
200
R59
2K
U306

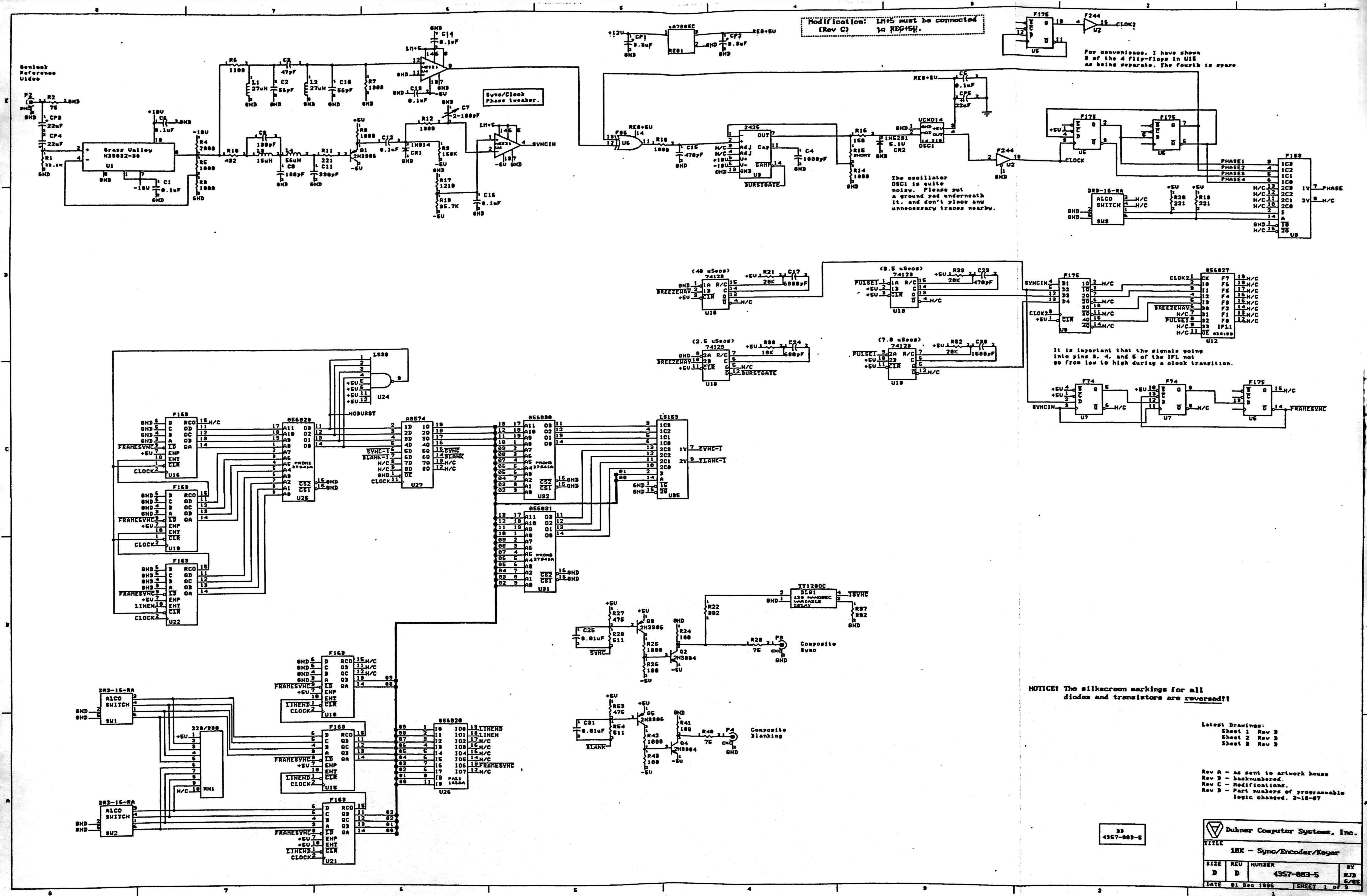
Dubner Computer Systems, Inc.

TITLE

10K - Display Plane

SIZE	REV	NUMBER	BY
D	A1 02/87	040015-00	RJB 10/84

DATE 02 Feb 1987 SHEET 6 of


Genlock Reference Video
Sync/Clock Phase tweaker.
Brass Valley H39832-38
U1
SYNCIN
uA7805C
REG+5V
CLOK2
CLOCK
PHASE1
PHASE2
PHASE3
PHASE4
PHASE
OSC1
BURSTGATE
BREEZEWAY
PULSET
FRAMESYNC
LINEND
HOBURST
SYNC-I
BLANK-I
ALCO SWITCH
DRD-16-RA
SW1
SW2
SW3
TT1280C
DL01
130 NANOSEC VARIABLE DELAY
TSYNC
Composite Sync
Composite Blanking
IFL1
PAL1
PROM1
PROM2
PROM3
RN1
33
4357-883-5

Modification: LM+5 must be connected
(Rev C) to REG+5V.

For convenience, I have shown
3 of the 4 flip-flops in U15
as being separate. The fourth is spare

The oscillator
OSC1 is quite
noisy. Please put
a ground pad underneath
it, and don't place any
unnecessary traces nearby.

It is important that the signals going
into pins 3, 4, and 5 of the IFL not
go from low to high during a clock transition.

NOTICE! The silkscreen markings for all
diodes and transistors are reversed!!

Latest Drawings:
Sheet 1 Rev D
Sheet 2 Rev D
Sheet 3 Rev D

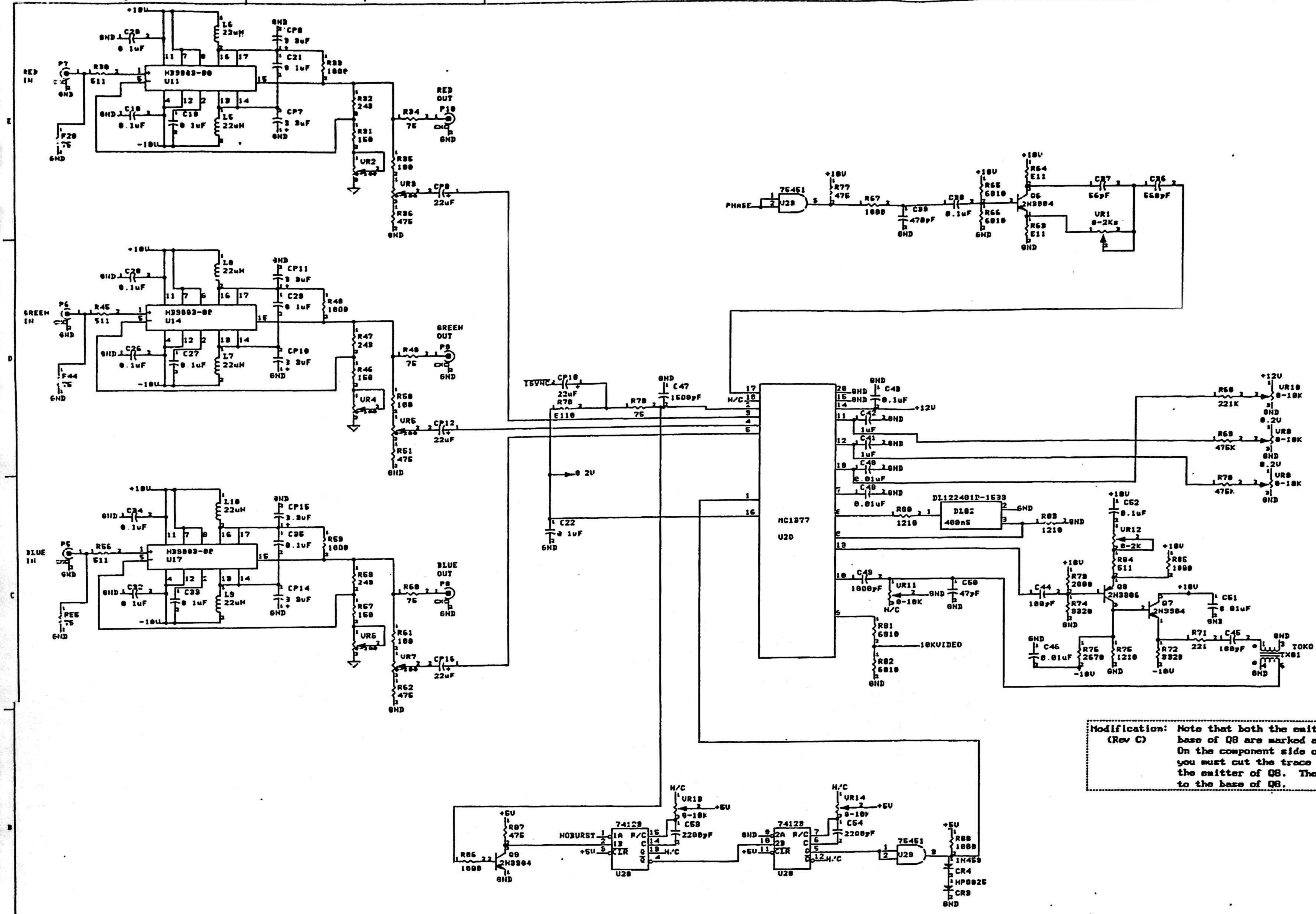
Rev A - as sent to artwork house
Rev B - backnumbered.
Rev C - Modifications.
Rev D - Part numbers of programmable
logic changed. 2-18-87

Dubner Computer Systems, Inc.

TITLE
10K - Sync/Encoder/Keyer

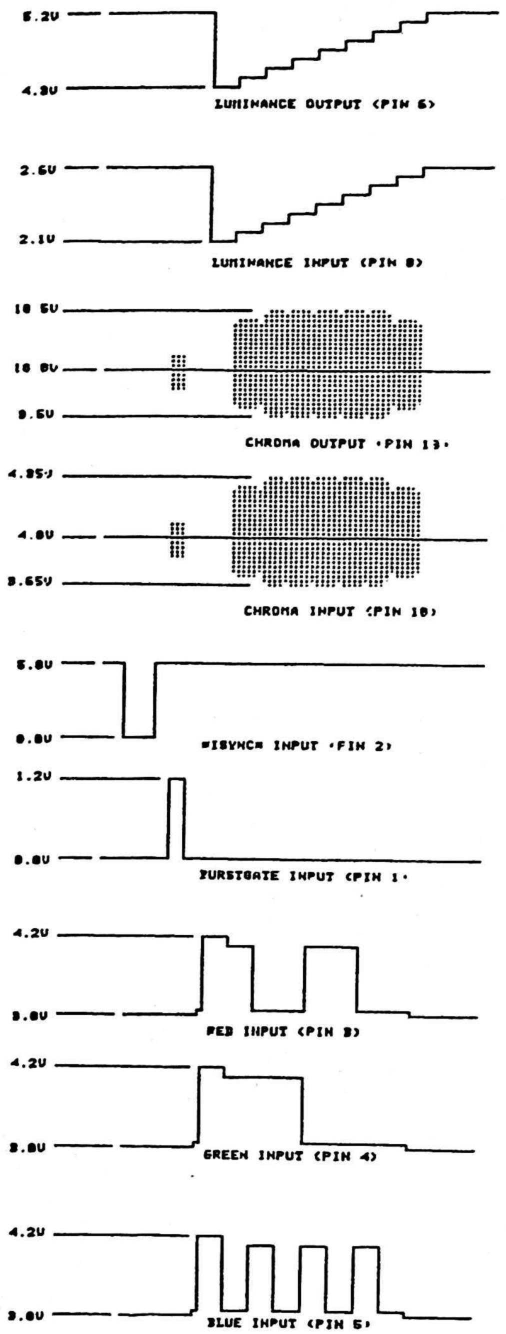
SIZE	REV	NUMBER	BY
D	D	4357-883-5	RJB 6/85

DATE 01 Dec 1986
SHEET 1 of 3


RED IN
P7
R38 511
U11 MJ9003-00
L5 22uH
L6 22uH
C21 0.1uF
R33 1000
R32 249
R31 150
VR2
R34 75
RED OUT
P10
VR3
CP9 22uF
R36 475
GREEN IN
P6
R45 511
U14
L8 22uH
L7 22uH
R48 1000
R47 249
R46 150
VR4
R49 75
GREEN OUT
P9
VR5
CP12 22uF
R51 475
BLUE IN
P5
R56 511
U17
L10 22uH
L9 22uH
R59 1000
R58 249
R57 150
VR6
R60 75
BLUE OUT
P8
VR7
CP15 22uF
R62 475
15VHC
CP10 22uF
R78 E110
R79 75
C47 1500pF
9.2V
C22 0.1uF
MC1377
U20
H/C
PHASE
75451
U29
R77 475
R67 1000
C39 470pF
C38 0.1uF
R65 6810
R66 6810
R64 E11
Q6 2N3904
R63 E11
C37 56pF
C36 560pF
VR1 0-2Ks
C43 0.1uF
+12V
C42 1uF
C41 1uF
C40 0.01uF
C48 0.01uF
DL1224010-1533
DL02 400nS
R80 1210
R83 1210
R68 221K
VR10 0-10K
R69 475K
VR8 0-10K
R70 475K
VR9 0-10K
C52 0.1uF
VR12 0-2K
R84 511
R85 1000
C49 1000pF
VR11 0-10K
C50 47pF
C44 100pF
R73 2000
R74 3320
Q8 2N3906
Q7 2N3904
C51 0.01uF
R71 221
C45 100pF
TOKO TX01
C46 0.01uF
R76 2670
R75 1210
R72 3320
R81 6810
R82 6810
10KVIDEO
R87 475
R86 1000
Q9 2N3904
HOBURST
74123
U28
VR13 0-10k
C53 2200pF
VR14 0-10k
C54 2200pF
75451
U29
R88 1000
1N459
CR4
HP0826
CR9
GND
+10V
-10V
+5V

Modification: (Rev C) Note that both the emitter and base of Q8 are marked as pin 1. On the component side of the board you must cut the trace from R74 to the emitter of Q8. Then jumper R74 to the base of Q8.

These waveforms should be seen on the indicated pins of the Motorola 1377

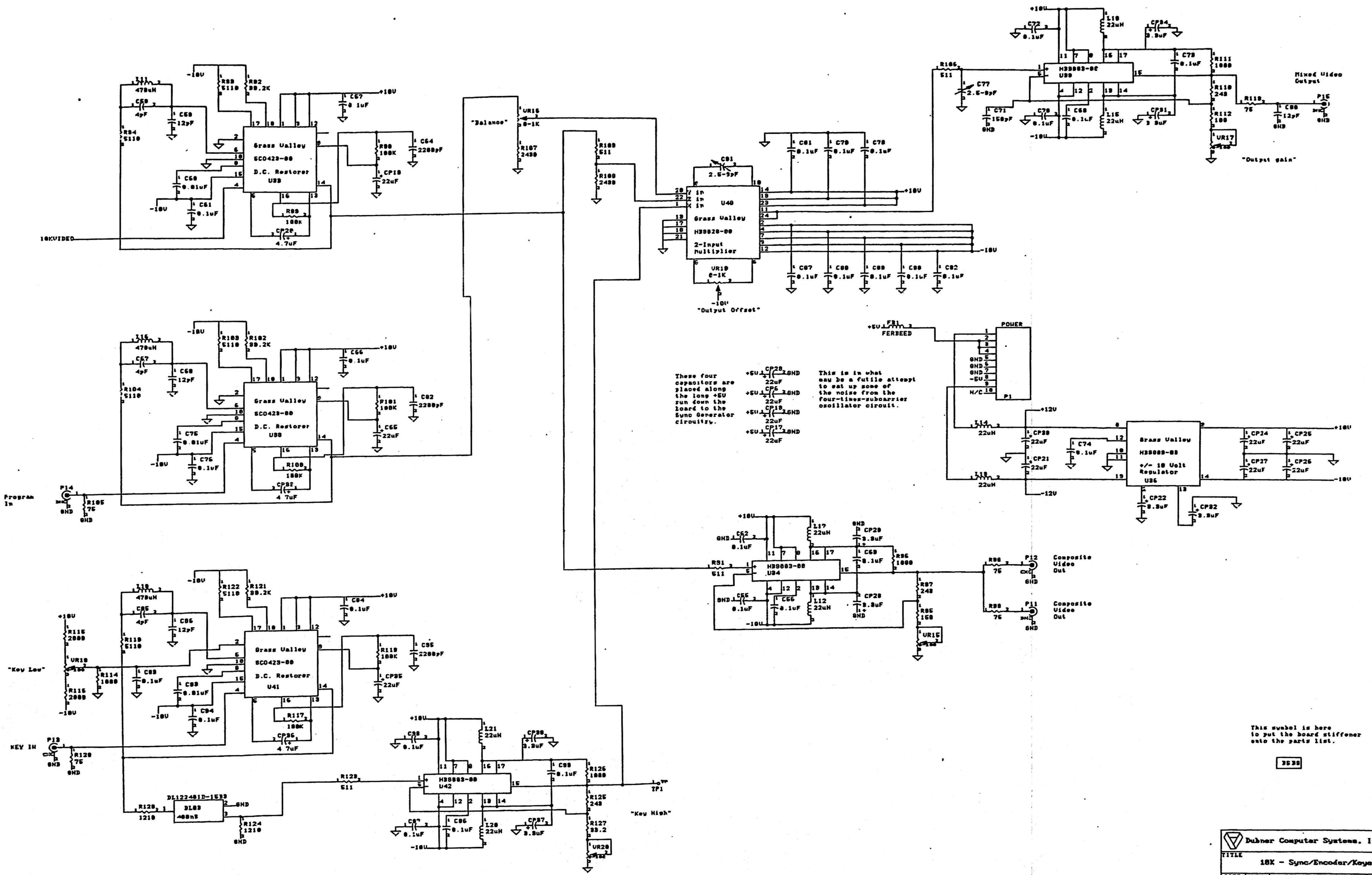


Dubner Computer Systems, Inc.

TITLE
10K - Sync/Encoder/Keyer

SIZE	REV	NUMBER	BY
D	D	4357-003-5	RJB 6/85

DATE 01 Dec 1986 SHEET 2 of 3


Grass Valley SC0423-00 D.C. Restorer U33
10KVIDEO
"Balance"
VR16 0-1K
U40 Grass Valley H30020-00 2-Input Multiplier
VR19 0-1K
"Output Offset"
H30003-02 U39
Mixed Video Output
P15
VR17
"Output gain"
Grass Valley SC0423-00 D.C. Restorer U38
Program In
P14
R105 75
POWER
P1
FB1 FERBEED
+5V
GND
-5V
N/C
Grass Valley H30009-03 +/- 10 Volt Regulator U36
+12V
-12V
+10V
-10V
Grass Valley SC0423-00 D.C. Restorer U41
"Key Lev"
VR18
KEY IN
P13
R120 75
H30003-00 U34
Composite Video Out
P12
P11
VR15
DL122481D-1533 DL83 400nS
H30003-00 U42
TP1
"Key High"
VR20
CP28 22uF
CP6 22uF
CP18 22uF
CP17 22uF
8
7
6
5
4
3
2
1
E
D
C
B
A

These four capacitors are placed along the long +5V run down the board to the Sync Generator circuitry.

This is in what may be a futile attempt to eat up some of the noise from the four-times-subcarrier oscillator circuit.

This symbol is here to put the board stiffener onto the parts list.

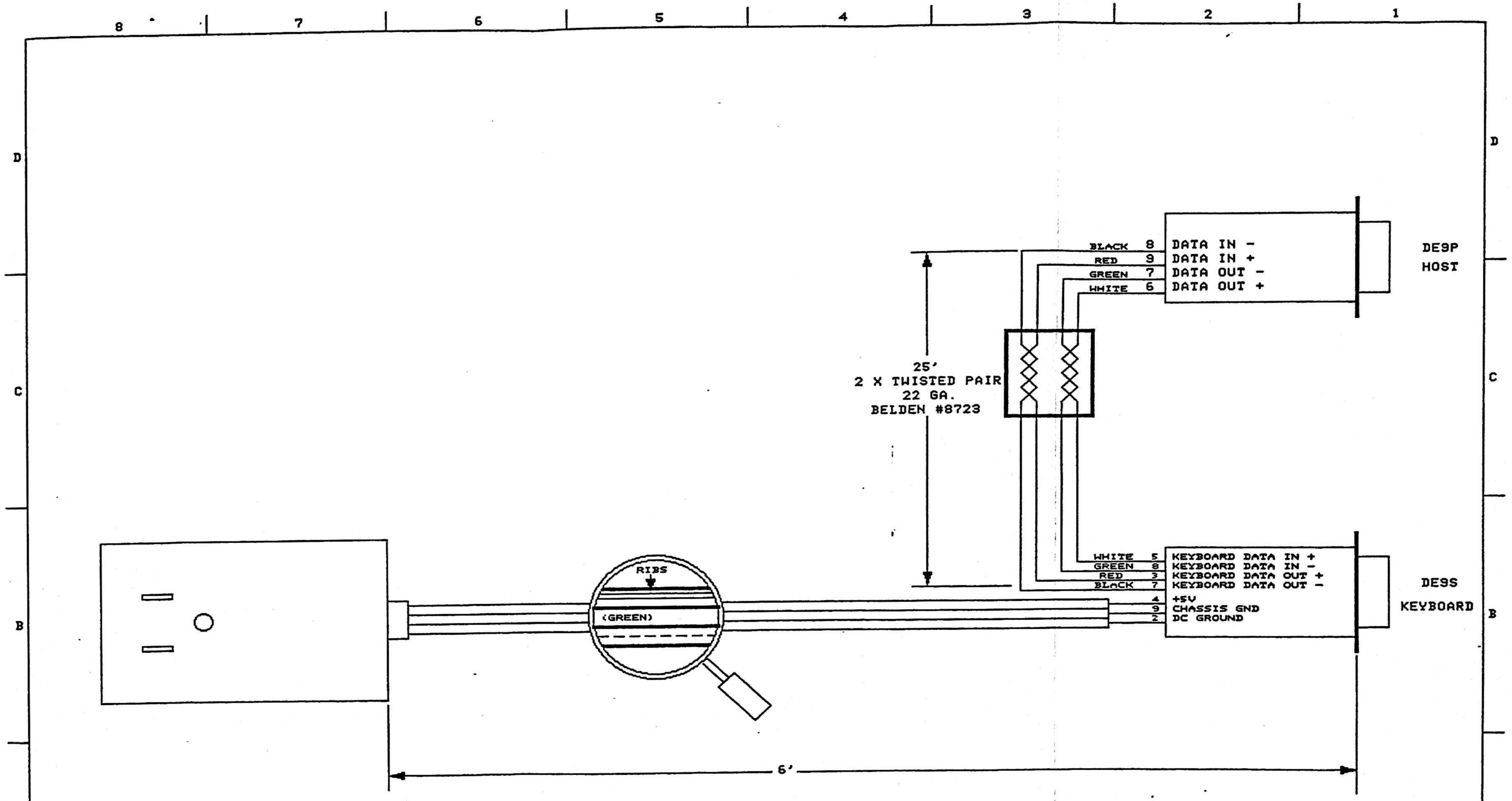
BSBS

Dubner Computer Systems, Inc.

TITLE
10K - Sync/Encoder/Keyer

SIZE	REV	NUMBER	BY
D	D	4357-003-5	RJB 10/84

DATE 01 Dec 1986 SHEET 3 of 8


8
7
6
5
4
3
2
1
D
C
B
A
BLACK 8 DATA IN −
RED 9 DATA IN +
GREEN 7 DATA OUT −
WHITE 6 DATA OUT +
DE9P
HOST
25'
2 X TWISTED PAIR
22 GA.
BELDEN #8723
WHITE 5 KEYBOARD DATA IN +
GREEN 8 KEYBOARD DATA IN −
RED 3 KEYBOARD DATA OUT +
BLACK 7 KEYBOARD DATA OUT −
4 +5V
9 CHASSIS GND
2 DC GROUND
DE9S
KEYBOARD
RIBS
(GREEN)
6'

DUBNER COMPUTER SYSTEMS, INC.

TITLE	Keyboard/ Call Up Box Cable w/ E.D.S Power Supply		
SIZE B	REV A	NUMBER 4357-008-2	BY BILL REHAK 2/86
DATE	27 APR 1987	SHEET 1 OF 1	